Серия института качества программного обеспечения

# БЫСТРОЕ ТЕСТИРОВАНИЕ

Роберт Капбертсон
Крис Браун
Гэри Кобб

Разработка программного обеспечения

# БЫСТРОЕ ТЕСТИРОВАНИЕ

*Ускорение тестирования сокращает срок выпуска продукта в свет!*

**Напряженная, жесткая, порою даже жестокая, конкуренция на современном рынке программного обеспечения заставляет максимально быстро поставлять программные продукты на рынок. Проблема непрерывного поддержания должного уровня качества и, в то же время, соблюдения календарного плана приводит к созданию новых стратегий тестирования, неразрывно связанных с процессом разработки. Несмотря на то, что идеальное программное обеспечение существовать не может, технология быстрого тестирования ставит своей целью выявление основных ошибок еще на ранних этапах процесса разработки, требуя тесной интеграции планирования, прогона тестов и документирования результатов на протяжении всего жизненного цикла.**

**В этой книге предложен подход, который позволяет модернизировать процесс тестирования и повысить качество программного обеспечения без дополнительных затрат времени. Книга состоит из трех частей:**

- **Принципы и процессы быстрого тестирования.**
- **Технологии, советы и реальные примеры.**
- **Пример полного набора документации по тестированию.**

**Делая акцент на скорости и тесной интеграции, материал этой книги покрывает весь жизненный цикл тестирования, включая:**

- **Планирование, разработку и реализацию тестовых случаев**
- **Отслеживание ошибок и построение отчетов по обнаруженным дефектам**
- **Технологии верификации и аттестации**
- **Метрики процесса тестирования и методы приближенных оценок необходимых ресурсов**
- **Выполнение измерений, создание отчетов, написание документации и осуществление пересмотров.**

## *Об авторах*

**Роберт Калбертсон** имеет за плечами 25-летний опыт в сфере разработки, тестирования и управления созданием программного обеспечения. Он работал во многих крупных компаниях, среди которых Cisco Systems, Texas Instruments, IBM, DSL Communications, а также в Техасском университете.

**Крис Браун** работает в компьютерной индустрии и индустрии программного обеспечения более 20 лет. Он сотрудничал со многими компаниями, в том числе Cisco Systems, Advanced Micro Devices, Compaq Computer Corporation, Dataserv/BellSouth и IBM.

**Гэри Кобб** преподает в институте качества программного обеспечения при Техасском университете в Остине, где он ранее учился на факультетах математики, информатики, электротехники и производства компьютеров. Ему довелось работать в нескольких крупных компаниях, среди которых Texas Instruments, Lockheed Martin и Dell Computer Corporation.

ISBN 5-8459-0336-X

PH PTR
PRENTICE HALL
Upper Saddle River, NJ 07458
www.phptr.com

ВИЛЬЯМС
Издательский дом "Вильямс"
www.williamspublishing.com

# Оглавление

# Содержание

# Предисловие

В данной книге дается описание практического подхода к тестированию программного обеспечения, при этом основное внимание уделяется процессам тестирования на фоне стремительного ускорения процесса разработки программного обеспечения. Этот подход ориентирован на применение специалистами по тестированию программного обеспечения и руководителями тестовых работ. При его описании приводится множество советов, технологий и примеров, которые могут использоваться для повышения эффективности и сокращения сроков тестирования программного обеспечения. Данная книга предназначена и для тех, кто решил начать свою служебную карьеру с тестирования программного обеспечения. Она содержит обширный набор ссылок, которые могут оказаться полезными как для уже сформировавшихся профессионалов, так и для тех, которые только начинают свою трудовую деятельность.

Скорость и эффективность разработки программного обеспечения зависят от того, насколько удачно процесс тестирования вписывается в общий жизненный цикл разработки программного продукта и от эффективности использования технологий тестирования. В книге показано, как увеличить скорость и эффективность тестирования, уделив основное внимание следующим вопросам:

- Начинать жизненный цикл тестирования необходимо одновременно с началом стадии формулирования технических требований, чтобы дефекты можно было обнаруживать как можно раньше и так же рано начинать планирование и реализацию тестовых случаев.
- Применять эффективные технологии статического тестирования, такие как инспекции и сквозной контроль, для тестирования промежуточных продуктов, которые создаются на протяжении жизненного цикла разработки.
- Применять эффективные технологии динамического тестирования для обнаружения дефектов на стадиях проверки взаимодействия и функционирования компонентов, системных и приемочных испытаний.

## Ключевые особенности

Повысить эффективность тестирования программного обеспечения помогут следующие отличительные особенности данной книги:

- Основное внимание уделяется настройке процесса тестирования так, чтобы достичь цели наискорейшего выхода на рынок при сохранении качества программного продукта.
- Тестирование программного обеспечения рассматривается в контексте общего жизненного цикла разработки программного обеспечения. Жизненный цикл разработки программного обеспечения рассматривается с точки зрения, выгодной для специалиста по тестированию. Рассматриваются также модели, такие как построение эволюционных прототипов, а также спиралевидная и традиционная каскадная модель.

- Представлены технологии статического тестирования, которые могут использоваться для подключения группы тестирования на ранних стадиях жизненного цикла разработки программного обеспечения. Применение статического тестирования позволяет обнаруживать дефекты на ранних стадиях жизни программного продукта и тогда же дает возможность составлять планы проведения испытаний и создавать тестовые случаи.
- Книга содержит примеры ключевых результатов процесса испытаний.

# Структура книги

Книга состоит из трех частей и организована следующим образом:

*Часть I. Процесс быстрого тестирования.* В этой части даны определения основных понятий и терминов, имеющих отношение к тестированию. В ней описаны процессы быстрого тестирования, которые тесно интегрированы с общим жизненным циклом разработки программного обеспечения. Рассматривается традиционная каскадная модель разработки программных продуктов, равно как и жизненные циклы, в основу которых положены инкрементальные поставки и построение эволюционно развивающихся прототипов. Каждая стадия процесса разработки программного продукта исследуется с точки зрения специалиста по тестированию, при этом дается описание методов обнаружения и предупреждения дефектов как средств повышения эффективности тестирования.

*Часть II. Технология быстрого тестирования и советы.* В этой части подробно описаны советы и технологии, которые могут быть использованы для внедрения процесса быстрого тестирования. Здесь представлены методы выявления и анализа технических требований, оценки трудозатрат на тестирование и разработки графиков выполнения тестовых работ, проведения инспекций и перепроверок, разработки тестов типа черного ящика. В этой части обсуждается широкий спектр технологий динамического тестирования, включая функциональный анализ, разбиение на классы эквивалентности, анализ граничных значений, проверка утечек памяти, тестирование случаев использования и характеристик производительности.

*Часть III. Примеры выполнения быстрого тестирования.* В третьей части приводятся примеры реализации процессов и технологий, которые обсуждались в двух предыдущих частях книги. В основе этих примеров лежит набор инструментальных средств управления тестированием (Test Management Toolkit, TMT), который является чисто учебным приложением. Продукт TMT позволяет руководителям тестовых работ и специалистам по тестированию осуществлять планы проведения испытаний, оглашать сообщения о дефектах, результаты тестирования и другую информацию, имеющую отношение к тестированию. В условиях Web-приложений появляется возможность одновременной работы нескольких пользователей над несколькими проектами независимо от их географического удаления друг от друга.

Существуют четыре рабочих продукта, имеющих отношение к процессу тестирования:

- Документ определения требований
- План тестирования
- Спецификация тестовой процедуры
- Сводный отчет о тестировании.

# Об авторах

**Роберт Калбертсон (Robert Culbertson)** имеет опыт работы в технических областях, в сфере разработки и тестирования программного обеспечения, а также опытом руководства разработкой различных проектов. Работая в компаниях Cisco Systems, Texas Instruments, IBM, в Техасском университете, в компании DSC Communications, он приобрел личный опыт работы с тем, что составляет быстрое тестирование. Роберт имеет ученые степени бакалавра электротехники и электроники (B.S.E.E.) и магистра электротехники и электроники (M.S.E.E.), полученные в Техасском университете в Остине, а также степень доктора философии в области электротехники и электроники от Бирмингемского университета в Англии.

**Гэри Кобб (Gary Cobb)** в течение 25 лет поднимался сразу по двум служебным лестницам, как преподаватель и как специалист, работающий в промышленной зоне города Остин. Он занимался преподавательской деятельностью в Техасском университете в Остине на отделениях математики, компьютерных наук, электротехники и производства компьютеров. Гэри также вел курс мультимедиа-систем на факультете компьютерных наук в Юго-западном университете штата Техас, в котором занимал пост руководителя лаборатории мультимедиа-систем. Его послужной список в промышленности включает работу на ответственных должностях в компаниях Texas Instruments Inc., Lockheed Martin и Dell Computer Corporation. Он читает краткие курсы в институте качества программного обеспечения при Техасском университете, который выступил спонсором его разработки критериев программного обеспечения, увенчавшейся присуждением ему премии Greater Austin Quality Award (Победитель по качеству в Остине). Гэри имеет степень доктора философии в области математики, полученную в Техасском университете в Остине.

**Крис Браун (Chris Brown)** работает в компьютерной индустрии и индустрии программного обеспечения более 20 лет. Занимаясь вопросами тестирования, он работал на различных должностях в таких компаниях, как Advanced Micro Devices, Cisco Systems, Compaq Computer Corporation и IBM. В компании Advanced Micro Devices он отвечал за системные испытания результатов генерации кремниевых компиляторов и за проверку на совместимость всех конфигураций аппаратных и программных средств. В компании Compaq Крис отвечал на построение тестовых сценариев и тестирование прототипов и опытных образцов систем для отделения портативных компьютерных продуктов. Будучи сотрудником компании IBM, Крис возглавлял команду тестирования администратора баз данных OS/2, администратора передачи данных и администратора локальной сети. Он был президентом/главным администратором в Computer Security Corporation, компании, которая поставляла программные средства защиты от вирусов военно-воздушным силам США, а также компаниям Prudential,

Boeing и другим крупным компаниям и лабораториям. Крис также занимал посты регионального менеджера и вице-президента в компании Dataserv Computer Maintenance/BellSouth, где отвечал за техническое обслуживание и ремонт в процессе эксплуатации 40000 компьютеров. Крис имеет степень бакалавра в области электроники и степень магистра по менеджменту.

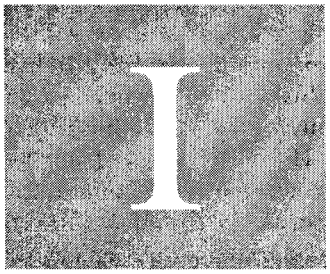
# Благодарности

Авторы остаются в неоплатном долгу перед Алом Дейлом (Al Dale), основателем института SQI (Software Quality Institute — институт качества программного обеспечения), за его поддержку, оказанную этой книге и серии в целом. Мы также хотели бы выразить свою благодарность Полу Петралия (Paul Petralia) за его моральную поддержку в период написания и издания этой книги. Наша признательность распространяется также и на влиятельных сотрудников института SQI и коллектива издательства Prentice Hall за их постоянную поддержку и профессиональную помощь. Джессика Болч (Jessica Balch) и весь производственный коллектив компании Pine Tree Composition проделали большую работу, чтобы вдохнуть в книгу жизнь; мы получили огромное удовольствие от сотрудничества с ними.

Роберт Калбертсон хочет выразить благодарность всей своей семье — Терри, Шилли и Майклу — они поддерживали своего папу тем, что не были ему помехой во время написания книги. Роберт благодарит свою маму Лорин и брата Лоренса за их неизменную помощь и поддержку. Роберт также выражает свою признательность всем менеджерам, инженерам и преподавателям, которые были для него одновременно и наставниками, и коллегами в течение многих лет. В частности, эти благодарности направлены в адрес Боба Маринконза (Bob Marinkonz), Марка Шервуда (Mark Sherwood) и многих других.

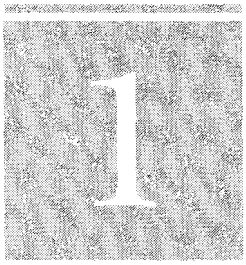
Гэри Кобб благодарит свою жену Мерилин за ее терпение и заботу в течение долгих часов, потребовавшихся для написания его части книги. Он также благодарит своих детей Гленна Кобба, Молли Пирсон, Стивена Кобба и Мередит Макларти за оказанную поддержку. В числе людей, оказавших ему помощь в служебном росте, Гэри прежде всего желает поблагодарить своих родителей, преподавателей и коллег по работе.

ЧАСТЬ



# Процесс быстрого тестирования

# Понятие о технологии быстрого тестирования




**Темы, рассматриваемые в главе:**

- Основные определения в области тестирования программного обеспечения
- Что такое быстрое тестирование?
- Разработка стратегии быстрого тестирования
- Процесс разработки программного обеспечения
- Каскадный процесс тестирования
- Связь тестирования и разработки
- Что дальше

За последних два десятилетия компьютерные системы и выполняемое на них программное обеспечение проникло во все области человеческой деятельности. Программное обеспечение присутствует в автомобилях, духовках, сотовых телефонах, играх и на рабочих местах. Оно является движущей силой бухгалтерских систем, систем обмена данными, соединений с Internet. Распространение программных систем достигло такой стадии, когда корпоративные и национальные экономики впадают во все большую зависимость от успешной разработки и доставки программного обеспечения.

По мере возрастания ставок на рынке программного обеспечения, все отчетливей вырисовывается потребность разработки большего числа программных продуктов за меньшие промежутки времени. Это обстоятельство предъявляет повышенные требования к разработчикам и тестировщикам программного обеспечения не только в плане ускоренного производства программных продуктов, но и в плане обеспечения должного уровня их качества, которое смогло бы удовлетворить потребителей.

В силу этого обстоятельства к специалистам по тестированию современного программного обеспечения предъявляются следующие основные требования:

- Необходимо тестировать быстро, соблюдая жесткие сроки поставки программных продуктов.

- Необходимо обеспечивать достаточно высокое качество тестирования, которое бы гарантировало, что дефекты, приводящие к разрушительным последствиям, не просочатся на компьютеры конечных пользователей.

Проблема связана с тем, чтобы удовлетворить каждое требование без ущерба для другого. Цель настоящей книги заключается в нахождении эффективного тестового процесса и представлении таких практических технологий, которые удовлетворяли бы обоим требованиям. Мы начнем с изучения фундаментальных основ разработки и тестирования программного обеспечения.

# Основные определения в области тестирования программного обеспечения

Прежде чем приступить к обсуждению процесса разработки программного обеспечения, дадим определения ряда основных терминов и понятий. По логике вещей лучше всего начать с определения, что собой представляет тестирование программного обеспечения.

*Тестирование программного обеспечения (software testing)*— это процесс анализа или эксплуатации программного обеспечения с целью выявления дефектов.

Несмотря на всю простоту этого определения, в нем содержатся пункты, которые требуют дальнейших пояснений. Слово *процесс (process)* используется для того, чтобы подчеркнуть, что тестирование суть плановая, упорядоченная деятельность. Этот момент очень важен, если мы заинтересованы в быстрой разработке, ибо хорошо продуманный, систематический подход быстрее приводит к обнаружению программных ошибок, чем плохо спланированное тестирование, к тому же проводимое в спешке.

Согласно этому определению, тестирование предусматривает "анализ" или "эксплуатацию" программного продукта. Тестовая деятельность, связанная с анализом результатов разработки программного обеспечения, называется *статическим тестированием (static testing)*. Статическое тестирование предусматривает проверку программных кодов, сквозной контроль и *проверку программы без запуска па машине, т.е. проверку за столом (desk checks)*. В отличие от этого, тестовая деятельность, предусматривающая эксплуатацию программного продукта, носит название *динамического тестирования (dynamic testing)*. Статическое и динамическое тестирование дополняют друг друга, и каждый из этих типов тестирования реализует собственный подход к выявлению ошибок.

Последний пункт определения, требующий дополнительных пояснений — это понятие *дефекта (bug)*. Говоря простыми словами, программная ошибка— ни что иное, как изъян в разработке программного продукта, который вызывает несоответствие ожидаемых результатов выполнения программного продукта и фактически полученных результатов. Дефект может возникнуть на стадии кодирования, на стадии формулирования требований или на стадии проектирования, либо же его причина может крыться в некорректной конфигурации или данных. Дефектом может быть также что-то другое, что не соответствует ожиданиям заказчика и что может быть, а может и не быть определено в спецификации программного продукта. Более подробное описание терминологии дефектов приводится во врезке 1.1.

**ДОЛГОВЕЧНОСТЬ ДЕФЕКТА**

Долговечность дефекта может быть описана следующим образом. Дефект возникает в результате того, что человек допускает *ошибку (error),* осуществляя некоторый вид деятельности, который имеет отношение к разработке программного обеспечения. К упомянутым видам деятельности относятся, например, формулирование требований, проектирование программы или написание программного кода. Эта ошибка вкрадывается в рабочий продукт (перечень требований, проектный документ или программный код) в виде *неисправности (fault).*

До тех пор пока эта неисправность (известная еще как *дефект (bug, defect))* присутствует в рабочем продукте, она может служить причиной появления других дефектов. Например, если неисправность, допущенная в перечне требований, остается необнаруженной, вполне вероятно, что это приведет к возникновению соответствующих ошибок в проекте системы, в проекте программы, в программном коде и даже в документации для пользователя.

Дефект остается необнаруженным до тех пор, пока не произойдет отказ. Ведь именно тогда пользователь или тестировщик замечает, что система не выполняет возложенных на нее функций. На стадии системных испытаний цель специалиста по тестированию состоит в том, чтобы вызвать сбой программы, обнаружить и задокументировать связанные с ним дефекты, а затем удалить их из системы. В идеальном случае долговечность дефекта ограничена моментом, когда он обнаруживается средствами статич*«-к<-.го или динамического тестирования и успешно устраняется.

*Врезка 1.1*

Одно из практических последствий определения процесса тестирования заключается в том, что перед специалистами по тестированию и разработчиками поставлены противоположные цели. Цель разработчика состоит в том, чтобы создать программный код без дефектов, который соответствует назначению программного продукта и отвечает требованиям заказчика. Разработчик пытается "сделать" программный код. Цель тестировщика связана с анализом кода и эксплуатацией программы, что в конечном итоге должно вести к вскрытию дефектов, затаившихся в программном коде, которые проявляются во время его интегрирования, конфигурирования и выполнения в различных средах. Тестировщик предпринимает попытки "разломать" программный код. В данном контексте хорошим результатом для разработчика **считается** успешное прохождение теста, в то время как для тестировщика успешный результат означает отказ программы на том же самом тесте. В конечном итоге и разработчик, и тестировщик стремятся к единой цели: получить такой программный продукт, который хорошо работает и удовлетворяет требованиям заказчика.

Тестирование программного обеспечения выполняет две базовых функции: верификацию и аттестацию. В [45] функции верификации и аттестации (verification and validation, V&V) определяются следующим образом:

*Верификация (verification)* обеспечивает соответствие результатов конкретной фазы процесса разработки требованиям данной и предшествующей стадий.

*Аттестация (validation)* есть гарантия того, что программный продукт удовлетворяет системным требованиям.

Цель аттестации заключается в том, что система должна отвечать всем предъявляемым требованиям, так чтобы происхождение каждой функции можно было проследить до конкретного требования заказчика. Другими словами, аттестация дает гарантию того, что строится правильный продукт.

Верификация в большей мере сосредоточена на действиях в рамках конкретной стадии процесса разработки. Например, одна из целей системного тестирования заключается в обеспечении соответствия проекта системы требованиям, которые были использованы как входные данные для стадии проектирования системы. Для подтверждения соответствия между проектом программы и проектом системы можно воспользоваться модульным тестированием и проверкой взаимодействия и функционирования компонентов системы. Проще говоря, верификация дает гарантию того, что продукт строится правильно. В последующих главах, параллельно с прохождением всех стадий процесса разработки, будут приводиться соответствующие примеры верификации и аттестации.

Остается дать определение еще одного дополнительного понятия — понятия качества. Подобно такому свойству, как красота, качество — во многом субъективное понятие, поэтому точное определение дать ему достаточно трудно. Определим качество программного обеспечения с использованием трех факторов: отказов на месте эксплуатации продукта, надежности и степени удовлетворенности заказчика. Говорят, что программный продукт обладает хорошим *качеством {quality),* если:

- При работе пользователя с программным продуктом возникает небольшое число отказов. Этот факт свидетельствует о том, что на рабочее место просочилось лишь небольшое число дефектов.
- Программный продукт надежен, а это означает, что его прогон редко завершается аварийным отказом или что он редко демонстрирует непредсказуемое поведение при работе в среде заказчика.
- Программный продукт удовлетворяет требованиям большинства пользователей.

Одно из следствий приведенного определения заключается в том, что тестовая группа не только должна принять меры к предотвращению и обнаружению дефектов в процессе разработки программного продукта, но также сконцентрировать свои усилия на повышении его надежности, удобства и простоты использования.

# Что такое быстрое тестирование?

В данной книге термин "быстрое тестирование" используется как дополнение понятия "быстрая разработка". Как отмечалось в [33], для разных людей быстрая разработка означает различные вещи. Некоторые понимают это как быстрое создание прототипов. Другие представляют ее как некоторое сочетание инструментальных средств CASE, активного участия пользователя и жестких временных ограничений. В редакции Макконнела [33] при определении быстрой разработки не упоминаются специальные инструментальные средства или методы:

> Быстрая разработка — это обобщенный термин, который означает то же, что "ускоренная разработка" и "сжатые сроки разработки". Он означает разработку программного продукта за меньшее время, чем это делалось до сих пор. Таким образом, "проект быстрой разработки" означает любой проект, при котором особое значение имеет срок разработки.

В том же ключе, *быстрое тестирование (rapid testing)* означает выполнение тестирования программного обеспечения в более быстром темпе, чем это делается в настоя-

щий момент, при условии сохранения или повышения уровня качества. К сожалению, простых путей достижения этой цели не существует. На рис. 1.1 показана упрощенная схема, представляющая быстрое тестирование как структуру, построенную на фундаменте из четырех компонентов. Если хотя бы один из этих компонентов ослаблен, эффективность тестирования существенно снижается. В соответствии с рис. 1.1, к четырем компонентам, которые должны быть оптимизированы для целей быстрого тестирования, относятся персонал, процесс комплексных испытаний, статическое тестирование и динамическое тестирование. Далее приводится краткий анализ всех четырех компонентов.

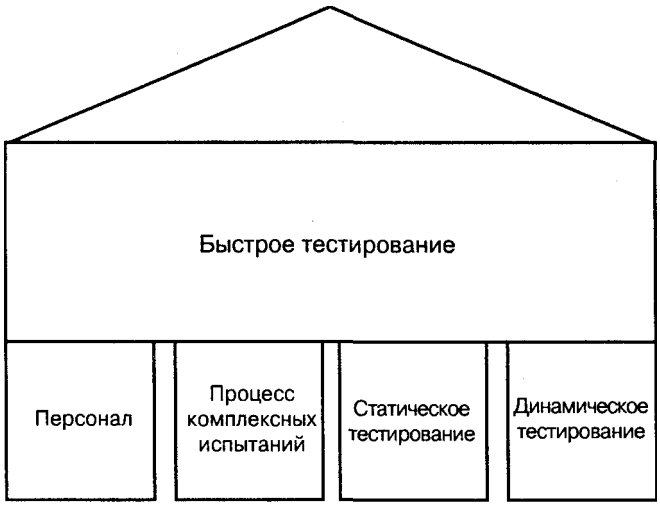



***Рис. 1.1. Наиболее важные компоненты быстрого тестирования***

## Персонал

Каждый менеджер по тестированию знает, что нужные специалисты представляют собой непременное условие быстрого тестирования. Многочисленные исследования показывают, что производительность разработчиков программного обеспечения различается в пределах 10:1 и даже больше. То же можно сказать и в отношении специалистов по тестированию — далеко не каждый специалист обладает навыками, опытом или характером, чтобы стать хорошим специалистом по тестированию. В частности, для выполнения быстрого тестирования нужны хорошо подготовленные и гибкие исполнители, способные работать в условиях жестких временных ограничений и быть полезными партнерами для разработчиков на ранних стадиях жизненного цикла разработки. Несмотря на то что в этой книге основное внимание уделяется процессу и технологии тестирования, в главе 6 предлагаются некоторые идеи, касающиеся персонала, который должен выполнять тестирование.

## Процесс комплексных испытаний

Независимо от того, насколько высока квалификация персонала, если они не располагают систематическим и отлаженным процессом тестирования, они не смогут работать с максимальной эффективностью. В основу процесса тестирования должны

быть положены устойчивые, фундаментальные принципы, а сам процесс тестирования должен быть тесно интегрирован с общим процессом разработки программного обеспечения. В части I этой книги большое внимание уделяется описанию способов совершенствования процесса тестирования. Вопросы применения практических технологий и рекомендации по реализации рассматриваются в части И. В центр внимания проводимого нами анализа попадает исследование вопросов более органичного интегрирования процесса разработки и тестирования.

## Статическое тестирование

В предыдущем разделе мы определили статическое тестирование как вид тестовой деятельности, связанной с анализом продуктов разработки программного обеспечения. Статическое тестирование проводится с целью *подтверждения* вывода о том, что рабочий продукт, например, спецификация проекта, правильно реализует все системные требования, и с целью *контроля* качества проекта. Статическое тестирование является одним из наиболее эффективных средств выявления дефектов на ранних стадиях разработки, благодаря чему достигается существенная экономия времени и затрат на разработку. Сюда входят проверки, сквозной контроль и экспертные оценки проектов, программных кодов и прочих рабочих продуктов, равно как и статический анализ с целью обнаружения дефектов в синтаксисе, структурах данных и других компонентах программного кода. Статическое тестирование по существу есть все, что можно сделать для выявления дефектов без прогона программного кода. Опыт, накопленный авторами книги, позволяет утверждать, что этим средством зачастую пренебрегают. Статическое тестирование будет предметом наших обсуждений на протяжении первых двух частей этой книги.

## Динамическое тестирование

Часто, когда специалисты думают о тестировании, они имеют в виду динамическое тестирование, в рамках которого предусматривается эксплуатация системы с целью выявления дефектов. Если статическое тестирование не предусматривает прогона программного продукта, то динамическое тестирование без такого прогона обойтись не может. Вообще говоря, динамическое тестирование состоит из прогона программы и сравнения ее фактического поведения с ожидаемым. Если фактическое поведение отличается от ожидаемого, это значит, что обнаружен дефект. Как будет показано в последующих главах, динамическое тестирование применяется для выполнения тестов различных типов, таких как функциональная проверка, испытания для определения рабочих характеристик и тестирование в предельных режимах. Динамическое тестирование является центральным звеном процесса тестирования программного обеспечения, и если планирование, проектирование, разработка и выполнение динамического тестирования выполнены недостаточно хорошо, то процесс тестирования не может быть эффективным. Динамическое тестирование выполняется не только силами тестовой группы; тестовая группа должна быть частью коллектива разработчиков и совместно с ними принимать участие в проверке взаимодействия и функционирования компонентов системы.

# Разработка стратегии быстрого тестирования

Если бы перед вами была поставлена задача исследовать текущий процесс разработки программного обеспечения с целью найти способы повышения эффективности тестирования, то какая часть процесса в первую очередь привлекла бы ваше внимание? Начнете ли вы эти исследования с анализа того, как планируется выполнять тестирование? С анализа средств и методов автоматизации разработки программного обеспечения? Что можно сказать об используемой вами системе отслеживания дефектов?

Принятый в этой книге подход предусматривает исследование каждой фазы процесса разработки программного обеспечения с позиций специалиста по тестированию. Основная цель заключается в определении возможных способов ускорения процесса тестирования при условии сохранения или даже повышения качества программного продукта. При этом возникает характерный образ специалиста по тестированию, который сидит на вращающемся стуле и поворачивается лицом к каждому аспекту процесса разработки, определяя, что можно сделать, дабы эта стадия процесса разработки не стала источником дефектов, либо выясняя, можно ли получить на этой стадии информацию, которая позволит ускорить процесс тестирования.

Прежде чем приступить к подробному анализу каждой стадии процесса разработки программного обеспечения в перспективе тестирования, потребуется заложить определенный фундамент. В этой главе даются определения основных терминов и понятий тестирования программного обеспечения. Затем проводятся исследования каждой фазы типового процесса разработки, которые имеют целью определить возможные способы ускорения процесса тестирования и повышения его эффективности. Во время исследования каждой стадии разработки мы рассчитываем получить ответы на следующие вопросы:

- Может ли тестовая группа предпринять на данной стадии какое-либо действие, способное предотвратить утечку дефектов?
- Может ли тестовая группа предпринять на данной стадии какое-либо действие, позволяющее уменьшить риск нарушения временного графика разработок?
- Можно ли получить какую-либо информацию на текущей стадии разработки, которая позволила бы тестовой группе ускорить планирование тестирования, разработку тестовых случаев или выполнение тестов?

Если тестовый процесс разработан с учетом ответов на эти вопросы, то это должно вызвать повышения темпов тестирования, равно как и качества конечного программного продукта.

# Процесс разработки программного обеспечения

До сих пор мы говорили об исследовании процесса разработки программного обеспечения, не указывая конкретно, что из этого следует. Одна из причин избегать однозначных утверждений заключается в том, что не существует процесса разработки, который наилучшим образом подходил бы для всех быстро разрабатываемых проектов. Например, встроенный контроллер для кардиостимулятора должен создаваться "максимально быстро", тогда как интерактивный словарь для вашего соседа вряд ли должен разрабатываться столь же быстрыми темпами.

**В** [42] *процесс (process)* определен как "последовательность шагов, в том числе действия, ограничения и ресурсы, которая приводит к желаемому результату определенного рода". Из предложенного Пфлигером [42] списка атрибутов процесса выделим следующие:

- В рамках процесса предварительно описываются все основные действия.
- В процессе задействуются ресурсы, с которыми связана некоторая совокупность ограничений (например, расписание), и генерируются промежуточные и конечные результаты.
- Процесс может состоять из некоторого числа подпроцессов, связанных между собой определенным образом. Процесс можно определить как некоторую иерархию процессов, организованных так, что каждый подпроцесс описывается собственной моделью процесса.
- С каждым действием процесса связаны критерии входа и выхода, так что известно, когда определенное действие начинается и когда завершается.
- Действия выполняются последовательно или параллельно по отношению к другим независимым подпроцессам, поэтому легко определить, когда некоторое действие выполняется относительно других действий.
- Каждый процесс управляется некоторым набором руководящих принципов, определяющих цели каждого действия.
- К действию, ресурсу или результату могут применяться ограничения или директивы. Например, бюджет или план накладывает ограничения на промежуток времени, в течение которого выполняется то или иное действие, а некоторое инструментальное средство может накладывать ограничения на способ использования определенного ресурса.

Когда процесс имеет отношение к созданию того или иного продукта, это часто называется жизненным циклом. По той же причине разработка программного продукта называется *жизненным циклом программного продукта (software life cycle)*. Жизненный цикл программного продукта может быть описан разными способами, в то же время для этой цели часто используется модель, которая представляет основные свойства процесса, сопровождаемые определенным сочетанием текста и иллюстраций.

Одной из первых была использована каскадная (или водопадная) модель жизненного цикла программного продукта, показанная на рис. 1.2. Основное свойство каскадной модели заключается в том, что каждая стадия или компонент модели завершается перед началом следующей стадии. Процесс начинается с определения требований к системе. В каскадной модели эти требования выявляются, анализируются и записываются в специальные документы еще до того, как начнутся работы по проектированию. Проектирование системы, проектирование программ, кодирование и различные виды тестирования суть самодостаточные и тщательно документированные стадии процесса. Следует, однако, заметить, что обычно некоторые стадии выступают под различными именами; например, стадия системного проектирования часто называется "проектным заданием" или "эскизным проектом", в то время на как стадию разработки программы зачастую ссылаются как на "рабочий план".

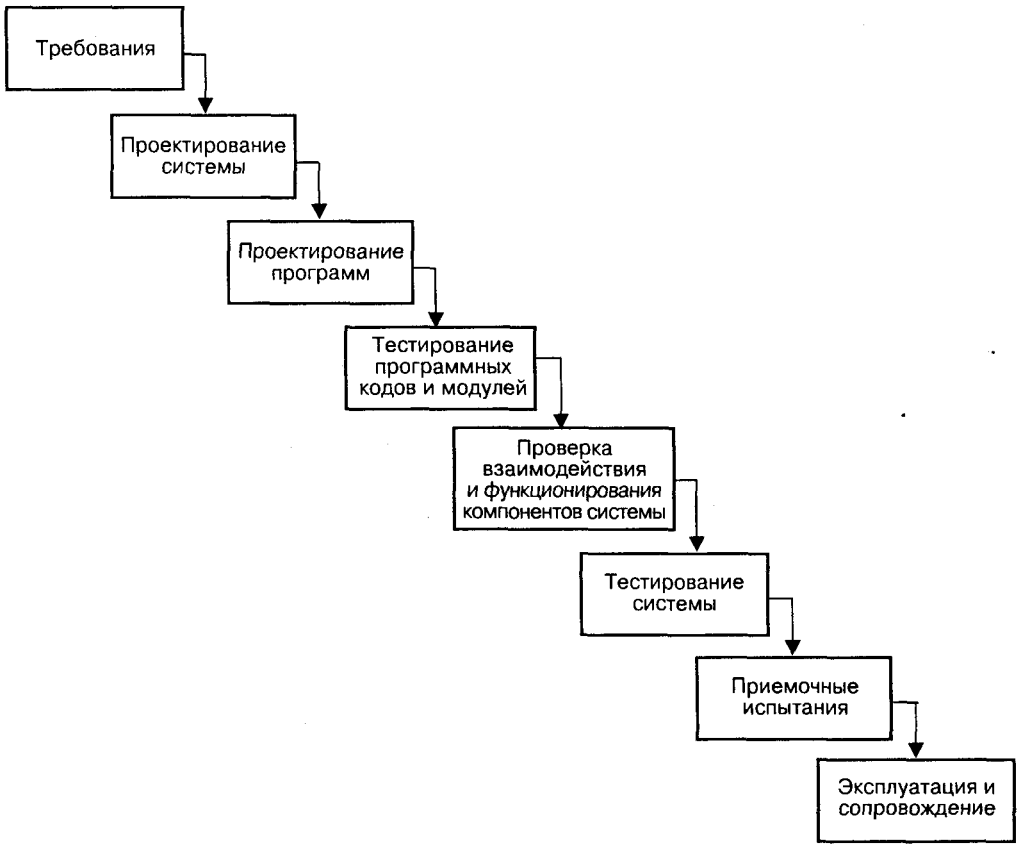



*Рис. 1.2. Каскадная модель жизненного цикла*

У каскадной модели имеются свои критики. Один из аргументов, к которым они прибегают, касается возможности фиксации всех требований на начальной стадии проекта. Предположим, что вас попросили высказать все свои требования к новому автомобилю до того, как он будет спроектирован и построен. Как заказчику, вам будет трудно сформулировать эти требования с той степенью детализации, которая необходима для проектирования и построения автомобиля "с нуля". Именно такие запросы предъявляет каскадная модель к заказчику и программисту-аналитику на начальном этапе каскадного процесса.

В [13] и [42] утверждается, что главным недостатком каскадной модели является то, что она не трактует программное обеспечение как процесс решения задачи. Каскадная модель заимствована из области разработки аппаратных средств. Она отображает конвейерный принцип разработки программного обеспечения, по условиям которого компонент сначала разрабатывается, а затем многократно тиражируется. Однако создание программного обеспечения — это, прежде всего, творческий, а отнюдь не производственный процесс. Становление программного обеспечения происходит по спирали по мере того, как растет понимание задачи. В рамках данного процесса происходят неоднократные продвижения вперед и возвраты обратно, при этом пробуются различные варианты с целью выбора из них наилучшего. Другими

словами, невозможно построить точную модель процесса разработки программного продукта в виде некоторого набора автономных стадий, а именно это и предполагает каскадная модель. Другие модели, в частности, спираль, поэтапная передача, эволюционирующая прототипная модель, гораздо лучше отражают итеративный характер разработки программного обеспечения. Итеративные модели более подробно рассматриваются в главе 2.

Если вы работали в качестве специалиста по тестированию в условиях каскадных моделей разработки, вы, скорее всего, имеете опыт решения другой задачи, которая довольно часто встречается в каскадных моделях. Если не предпринять специальных предосторожностей, все ошибки, допущенные при формулировании требований, при проектировании системы и при написании программных кодов перетекают в организацию тестов. В случае применения каскадной модели тестовая группа может обнаружить массу дефектов перед самым окончанием разработок — дефектов, возникновение которых прослеживается вплоть до стадий формулирования требований, проектирования или кодирования профаммного продукта. Возврат к началу каскадного процесса разработки сопряжен с существенными трудностями и большими затратами времени и средств, поскольку все рабочие продукты, которые якобы уже прошли завершающие стадии, должны подвергнуться повторной проверке.

Несмотря на все порождаемые проблемы, каскадная модель достойна того, чтобы ее изучить, поскольку она содержит основные компоненты, необходимые для разработки программных продуктов. Независимо от используемой модели, разработка программного обеспечения должна начинаться с понимания того, *что* нужно построить, другими словами, с выявления и анализа требований. Процесс разработки должен включать проектирование, кодирование и тестирование невзирая на то, выполняются ли они в виде линейной последовательности, что характерно для каскадной модели, или в рамках итеративной последовательности, характерной для моделей эволюционного прототипирования и поэтапной передачи. Мы используем каскадную модель в качестве контекста для обсуждения, как добиться улучшения процесса, однако базовые принципы быстрого тестирования должны применяться независимо от выбранной модели жизненного цикла.

## Каскадный процесс тестирования

В традиционной каскадной модели (см. рис. 1.2) роль организации тестов остается неясной до стадий системного тестирования и приемочных испытаний. Большая часть видов деятельности, характерных для ранних стадий, таких как проектирование, кодирование и модульное тестирование, в первую очередь связано с коллективом разработчиков программного обеспечения. По этой причине имеет смысл построить соответствующую модель жизненного цикла процесса тестирования. Из того, что разработка динамических тестов является процессом, во многом совпадающим с процессом разработки программного обеспечения, следует, что каскадная модель жизненного цикла динамического тестирования довольно сильно похожа на модель, которая обсуждалась в предыдущем разделе. Пример каскадной модели жизненного цикла процесса тестирования приводится на рис. 1.3. Обратите внимание на тот факт, что данная модель описывает динамическое тестирование, однако в ней не отражено статическое тестирование.

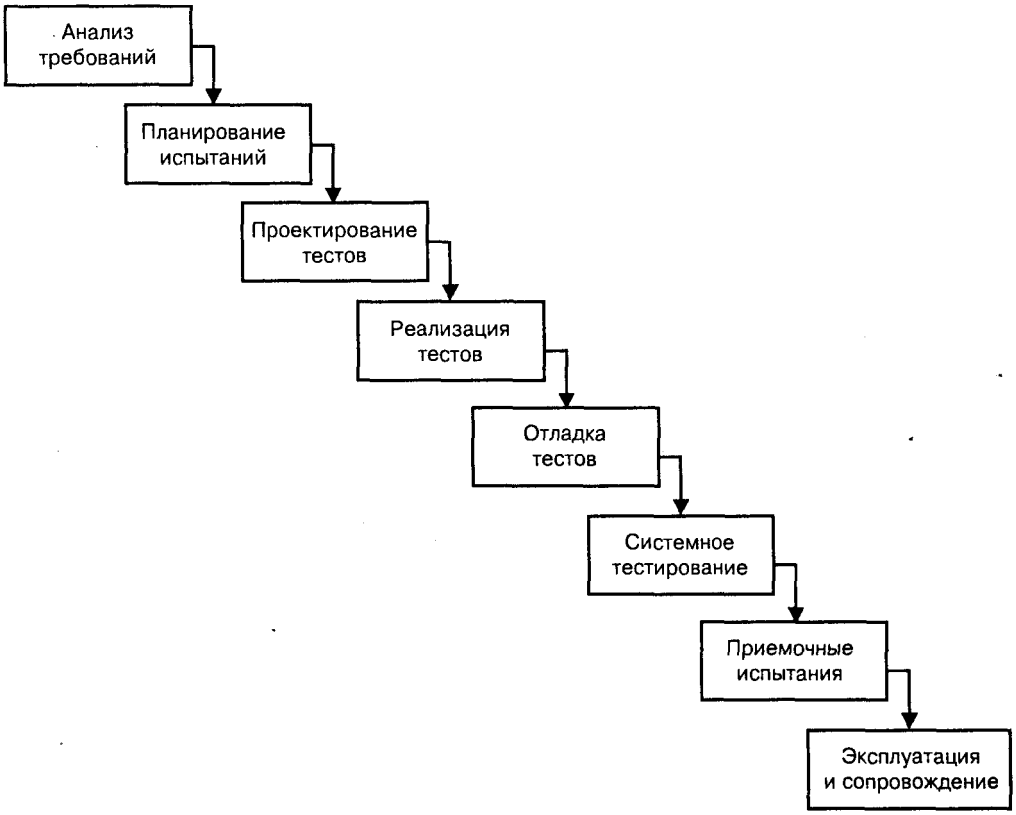



***Рис. 1.3. Каскадный процесс тестирования***

Сводка входных и выходных данных для каждой стадии каскадного процесса отладки представлена в таблице 1.1. Далее будет дано краткое описание действий, входов и выходов для каждой стадии каскадного процесса тестирования. Другие подробности рассматриваются в оставшихся главах первой части.

## Анализ требований

На этапе анализа требований цели, которые преследует группа специалистов по тестированию и коллектив разработчиков, различаются в ряде аспектов. Обоим коллективам нужны четкие, недвусмысленные спецификации требований, которые служат входными данными для их работы. Разработчики желают получить полный набор требований, которыми можно воспользоваться для формулирования функциональных требований к системе и которые позволили бы проводить работы по проектированию и кодированию программного продукта. С другой стороны, тестовой группе нужен набор требований, который позволил бы им составить план проведения испытаний и выполнить системные и приемочные испытания.

Таблица 1.1. Входные и выходные данные для каскадного процесса тестирования

| Вид деятельности | Входы | Выходы |
|---|---|---|
| Анализ требований | Определение требований, спецификация требований | Матрица прослеживаемости требований |
| Планирование испытаний | Спецификация требований, матрица прослеживаемости требований | План проведения испытаний — стратегия тестирования, испытательная система, оценка объема работ и расписание тестирования |
| Проектирование тестов | Спецификация требований, матрица прослеживаемости требований, план проведения испытаний | Проектирование тестов — спецификация входных данных тестов, конфигурации тестов |
| Реализация тестов | Функциональная спецификация программного обеспечения, матрица прослеживаемости требований, план проведения испытаний, проекты тестов | Тестовые случаи — методы проверки и автоматизированные тесты |
| Отладка тестов | "Ранний вариант" кода, тестовые случаи, рабочая испытательная система | Тестовые случаи для заключительных испытаний |
| Системное тестирование | План испытания системы, матрица прослеживаемости требований, завершающие тестовые случаи, рабочая испытательная система | Результаты тестирования — сообщения о дефектах, сообщения о состоянии испытаний, суммарный отчет о результатах тестирования |
| Приемочные испытания | План приемочных испытаний, матрица прослеживаемости требований, бета-вариант программного кода, тестовые случаи для приемочных испытаний, рабочая испытательная система | Результаты тестирования |
| Эксплуатация и сопровождение | Скорректированный код, тестовые случаи для контроля дефектов, регрессионные тестовые случаи, рабочая испытательная система | Местоположение обнаруженных дефектов |

Очень полезным выходным документом стадии анализа требований, как для разработчиков, так и для тестовой группы, является матрица прослеживаемости требований. *Матрица прослеживаемости требований (requirements traceability matrix)* представляет собой документ, который отображает каждое требование на промежуточные результаты процесса разработки, такие как компоненты проекта, модули программного обеспечения и результаты тестирования. Она может быть представлена в виде электронной таблицы, таблицы текстового процессора, базы данных или Web-страницы. Матрица прослеживаемости требований и ее роль в "соединении" различных видов деятельности процессов разработки и тестирования подробно рассматривается в главе 2.

## Планирование испытаний

Под планированием испытаний понимается определение объемов испытаний, подходов, ресурсов и расписания выполнения намеченных действий. Для того чтобы тестирование было эффективным, необходимо потратить значительные средства и усилия па планирование, а также быть готовым пересматривать этот план в динамике, соответственно изменениям в требованиях, расчетах или программных кодах по мере выявления дефектов. Очень важно, чтобы все требования подверглись тестированию, либо же, если требования выстроены по некоторой системе приоритетов, по крайней мере, должны быть проверены требования с максимальными приоритетами. Матрица прослеживаемости требований является полезным инструментальным средством на стадии планирования испытаний, поскольку она может использоваться при расчете объемов тестирования, необходимого для охвата важнейших требований.

В идеальном случае планирование испытаний должно принимать в расчет как статическое, так и динамическое тестирование, но поскольку процесс тестирования, отраженный на рис. 1.3 и в таблице 1.1, отдает предпочтение динамическому тестированию, временно оставим статическое тестирование в покое. Действия, выполняемые на стадии планирования испытаний, представляют собой подготовительные этапы для этапов системных и приемочных испытаний, которые расположены ближе к концу каскада, и должны включать:

- Определение того, что подлежит тестированию, и подхода, который при этом будет использоваться.
- Отображение тестов на требования.
- Определение критериев входа и выхода для каждой стадии процесса тестирования.
- Оценка персонала, необходимого для выполнения тестовых работ, по квалификации и степени занятости.
- Оценка времени, необходимого для выполнения работ по тестированию.
- Планирование основных этапов работ.
- Определение тестовой системы (аппаратных и программных средства), необходимой для проведения тестирования.
- Определение рабочих продуктов для каждой фазы тестирования.
- Оценка рисков, связанных с тестированием, и план по их уменьшению.

Промежуточные результаты или выходы, являющиеся результатами этих действий, могут быть включены в план проведения испытаний, который, как правило, состоит из одного или большего числа документов. Более подробно планирование испытаний будет рассматриваться в главе 3, а с примером плана проведения испытаний можно ознакомиться в третьей части книги.

## Проектирование тестов, реализации и отладка

Динамическое тестирование основано на выполнении заданного набора операций на конкретном модуле программного продукта и сравнении фактически полученных результатов с ожидаемыми. Если после прогона получен ожидаемый результат, счи-

тается, что модуль прошел тест. Если зафиксировано аномальное поведение, тест считается неудачным, однако он может оказаться успешным в смысле обнаружения дефекта. Заданный набор выполняемых операций образует тестовый случай. Следует подчеркнуть, что тестовые случаи должны быть спроектированы, закодированы и отлажены до того, как их можно будет использовать.

Проектирование тестов состоит из двух компонентов: архитектуры тестов и подробных планов тестов. Архитектура тестов упорядочивает тесты по группам, таким как функциональные тесты, испытания для определения рабочих характеристик, проверка безопасности и т.д. Она также описывает структуру и соглашения по именованию хранилища тестов. Подробные планы тестов описывают назначение каждого теста, технические средства и данные, необходимые для выполнения теста, ожидаемые результаты каждого теста, а также указывают на требование, на подтверждение выполнения которого ориентируется данных тест. Между требованиями и планами тестов должно существовать, по меньшей мере, отношение один к одному.

На основе планов тестов могут быть разработаны подробные методики проверки. Уровень детализации, необходимый для представления методики проверки в виде письменного документа, зависит от квалификации и уровня знаний персонала, выполняющего прогон тестов. Следует найти компромисс между временем, необходимым для написания подробной, последовательной методики, и временем, которое заинтересованное лицо тратит на обучение правильному выполнению тестов. Даже если тест должен быть автоматизирован, в общем случае затраты времени на заблаговременное подробное описание методики проверки оправдываются, поскольку перед специалистом по автоматизации ставится четкая и однозначная задача автоматизации теста.

Как только методика тестирования будет описана, ее потребуется проверить на каком-либо модуле программного продукта. Поскольку, скорее всего, этот тест будет проверяться на "изобилующей ошибками" программе, особо внимательно следует анализировать случаи, когда тест не проходит, что позволит определить, где находится причина неудачи — в тестируемом коде или в самом тесте.

## Системное тестирование

Набор завершенных, отлаженных тестов может использоваться и на следующей стадии каскадного процесса тестирования, т.е. на этапе системных испытаний. Системное тестирование проводится для удостоверения того, что программное обеспечение делает именно то, что от него ожидает пользователь. Существуют два основных типа системных испытаний: функциональная проверка и испытания для определения рабочих характеристик.

*Функциональная проверка (functional testing)* не требует от тестировщика знаний принципов работы программного продукта, в то же время она требует знаний функциональных требований, предъявляемых к системе. Она использует набор тестов, которые определяют, выполняет ли система все то, что она должна делать с точки зрения пользователя.

После того, как тестирование подтвердило адекватность базовой функциональности системы, задачей тестирования становится проверка, насколько хорошо система выполняет свои функции. В рамках *испытаний для определения рабочих характеристик (performance testing)* выполняются такие проверки, как тестирование в предельных ре-

жимах, нагрузочные испытания, контроль синхронизации и проверка восстанавливаемости. Испытания на надежность, на эксплуатационную готовность, проверка приспособленности к техническому обслуживанию также можно включить в число испытаний для определения рабочих характеристик.

Помимо функциональной проверки и испытаний для определения рабочих характеристик, возможны различные дополнительные проверки, проведение которых может понадобиться на стадии системных испытаний. В их число могут входить проверка безопасности, установочная проверка, проверка на совместимость, проверка удобства и простоты использования, проверка возможностей наращивания. Более подробно системное тестирование рассматривается в главе 5 и в главах второй части.

## Приемочные испытания

По завершении системного тестирования продукт может быть передан пользователю для проведения приемочных испытаний. Если пользователи принадлежат той же компании, что и разработчики, такое тестирование обычно называется *альфа-тестированием (alpha testing)*. Если пользователями являются заказчики, готовые работать с программным продуктом еще до его официальной готовности, то такое тестирование называется *бета-тестированием (beta testing)*. И альфа-, и бета-тестирование представляют собой соответствующие формы *контрольных испытаний (pilot tests),* в условиях которых система устанавливается на экспериментальной базе с целью выявления дефектов.

Другой формой приемочных испытаний являются *аттестационные испытания (benchmark test),* когда заказчик выполняет заранее определенный набор тестовых случаев, имитирующих типовые условия, в которых система будет работать после ввода в эксплуатацию. Для аттестационных испытаний могут быть использованы тестовые случаи, которые разработаны и отлажены вашей тестирующей организацией и которые в то же время были проверены и одобрены заказчиком. По завершении контрольных и аттестационных испытаний заказчик должен уведомить вас, какие требования не удовлетворены, а какие должны быть изменены, чтобы можно было перейти к заключительным испытаниям.

Заключительным типом приемочных испытаний является *установочная проверка (installation test),* по условиям которой завершенная версия программного продукта устанавливается на площадках заказчика с целью получить от него подтверждение, что программный продукт соответствует всем требованиям и заказчик согласен на его поставку.

## Сопровождение

Сопровождение программного продукта часто ставит разработчиков и тестировщиков перед необходимостью решения определенных задач. Сопровождения для разработчика — это исправление дефектов, которые были обнаружены заказчиком во время эксплуатации программного продукта, а также расширение функциональных возможностей продукта, которое призвано удовлетворить возросшие требования заказчика. Что касается организации тестирования, то сопровождение означает проверку результатов исправления дефектов, тестирование расширенных функциональных возможностей и выполнение *регрессионных тестов (regression tests)* на новых версиях

программного продукта. Цель всего этого заключается в получении подтверждения того, что ранее исправно работавшие функциональные средства не пострадали от внесения изменений в программный продукт.

Какими бы ни были важными приемочные испытания и сопровождение программного продукта, в данной книге они подробно не обсуждаются. Базовые принципы регрессионного тестирования и верификации дефектов хорошо вписываются в эти фазы жизненного цикла. За подробным анализом сопровождения программного обеспечения в перспективе тестирования рекомендуем обратиться к [30].

## Связь тестирования и разработки

В нескольких предыдущих разделах были описаны каскадные модели процесса разработки программного обеспечения и процесса тестирования. У обоих моделей общие исходные и конечные точки, однако группы разработчиков и тестировщиков все время заняты различными видами деятельности. В этом разделе даны описания двух моделей, которые связывают оба эти вида деятельности воедино.

Один из способов продемонстрировать, как соотносятся тестирование и разработка, показан на V-образной диаграмме, изображенной на рис. 1.4. В этой модели, на которую также ссылаются как на шарнирно-каскадную, оба вида деятельности, анализ и проектирование, образуют левую сторону V. Кодирование находится в самой нижней точке диаграммы, а тестирование образует ее правую сторону. Для простоты изложения, вид деятельности, подобный сопровождению, на диаграмме не показан.

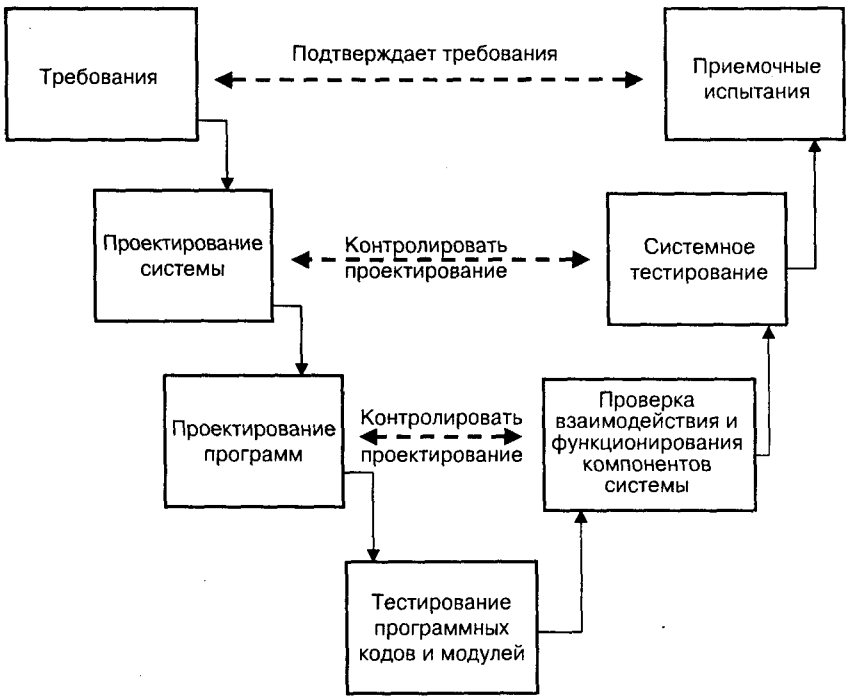



*Рис. 1.4. Шарнирно-каскадная, или V-диаграмма*

Пунктирные линии со стрелками на концах определяют отношения между видом тестовой деятельности на правой стороне и видом проектной деятельности на левой. Самая верхняя пунктирная линия со стрелками показывает, что цель приемочных испытаний заключается в подтверждении требований, а сами приемочные испытания основаны на требованиях. Аналогично, системные испытания служат для проверки проекта системы, а системные тесты получены по результатам проектирования системы. Следовательно, одно из назначений V-диаграммы состоит в том, чтобы показать, какова цель видов тестовой деятельности в терминах контроля и аттестации на ранних стадиях разработки.

Несмотря на то что V-диаграммы служат иллюстрацией отношений, связывающих разработку и тестирование, она не отражает двух параллельных потоков деятельности, которые две эти группы специалистов осуществляют в процессе разработки. Иллюстрацией одного из способов показать отдельные действия, связанные с разработкой и тестированием, служит параллельная каскадная модель на рис. 1.5.

Эта диаграмма несколько сложнее предыдущих моделей, однако, она обладает несколькими важными свойствами, которые оправдывают наличие дополнительной сложности.

- Отчетливо видны параллельные потоки разработки и тестирования.
- Стрелка, соединяющая системное тестирование и проектирование системы, говорит о том, что цель стадии системного тестирования заключается в проверке проекта системы.
- Стрелка, соединяющая приемочные испытания с требованиями, показывает, что назначение приемочных испытаний заключается в подтверждении требований.
- Аналогичные стрелки говорят о том, что назначение тестирования модулей и проверки взаимодействия и функционирования компонентов системы заключается в контроле разработки программы, а цель отладки тестов состоит в контроле разработки тестов.
- Каждая оставшаяся стадия потоков разработки и тестирования имеет "петли обратной связи", назначение которых заключается в проверке того, что промежуточные результаты системного проектирования, проектирования программ, планирования испытаний и тому подобного, соответствуют требованиям, предъявленных к ним в начале соответствующей стадии. Такая проверка выполняется в рамках статического тестирования.

Несложный анализ рис. 1.5 показывает, почему тестирование программного обеспечения является столь ответственной и трудоемкой задачей. Даже при работе в традиционной каскадной среде группа специалистов по отладке должна уметь выполнять широкий спектр действий, каждое из которых зависит от результатов деятельности коллектива разработчиков. Между потоками разработки и тестирования этого процесса должен надежно поддерживаться обмен данными, и каждая группа должна участвовать в некоторых контрольных действиях другой группы. Например, группа тестировщиков должна участвовать в контроле системного проектирования, а группа разработчиков — принимать участие в проверке плана тестирования.

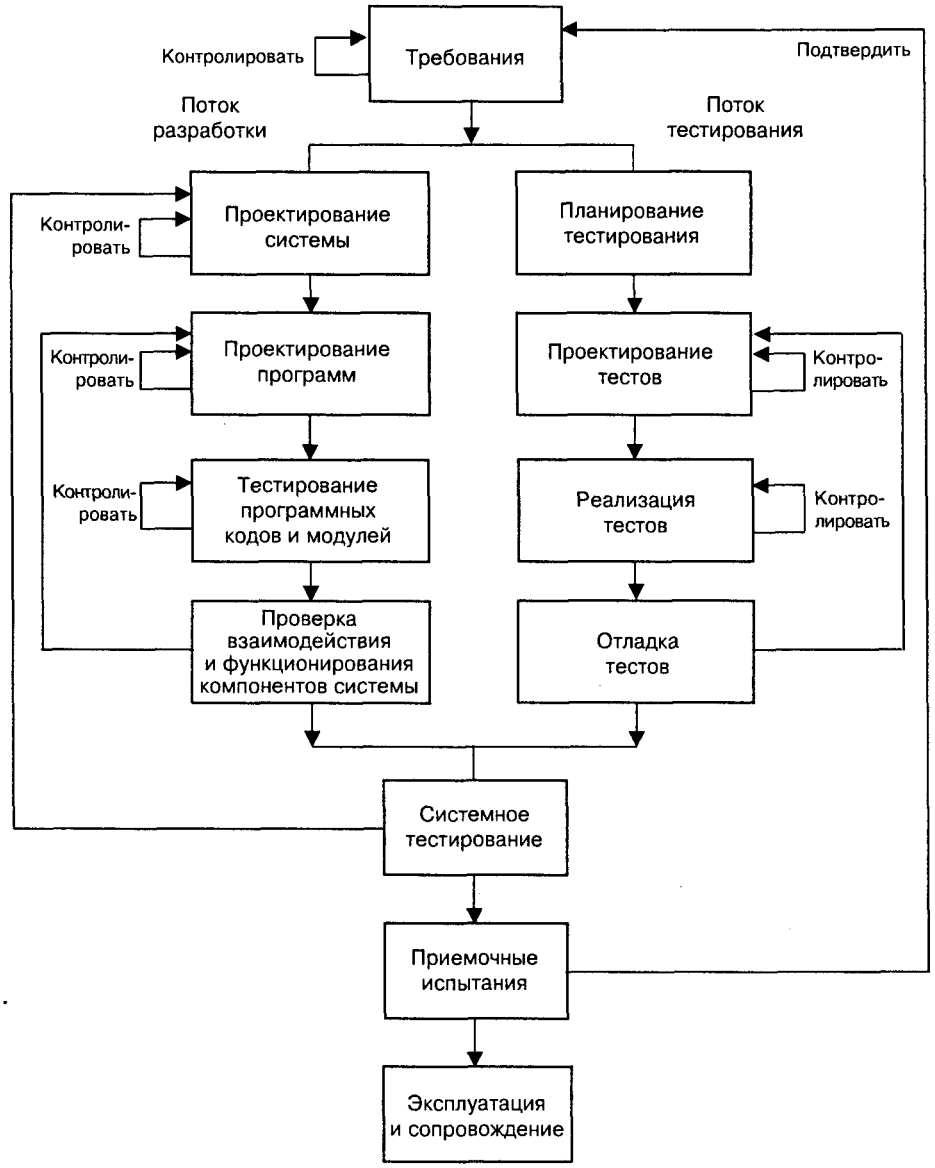


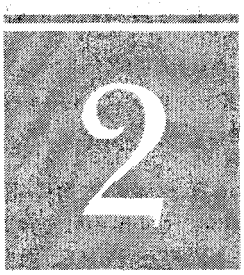

*Рис. 1.5. Параллельная каскадная модель*

## Что дальше

В этой главе были даны определения основных понятий тестирования программного обеспечения. Идея быстрого тестирования представлена как способ ускорения тестирования без ущерба для качества программного продукта. Мы установили, насколько важна роль персонала, собственно процесса, статического тестирования и динамического тестирования для организации эффективного процесса быстрой отладки. Были исследованы каскадный процесс разработки программного обеспечения и динамический процесс тестирования, и в результате обнаружено, что оба они должны быть тесно интефированы, если мы заботимся о высокой эффективности работ по тестированию. Наконец, были рассмотрены V-диаграмма и параллельная каскадная модель как средства, связующие воедино процессы разработки и тестиро. вания.

Интеграция процессов тестирования и разработки была начата с построения параллельной каскадной модели, однако впереди еще предстоит проделать большую работу, прежде чем определение процесса быстрого тестирования будет окончательно сформулировано. Потребуется проанализировать каждую стадию процесса тестирования на предмет возможной оптимизации ее скорости и эффективности. Эта работа будет начата в следующей главе.

# Анализ требований и тестирование




**Темы, рассматриваемые в главе:**

- Процесс формулирования требований
- Тестирование требований
- Что дальше

В предыдущей главе мы установили, что в целях ускорения производства программного продукта, разработка и все виды тестовой деятельности должны тесно интегрироваться. Такая интеграция разработки и тестовой деятельности должна начинаться на ранних стадиях процесса разработки, когда формулируются требования к разрабатываемому программному продукту при непосредственном участии пользователя. Для проектирования системы программного обеспечения коллективу разработчиков необходим четкий набор требований, в то же время группе специалистов по отладке также необходимы четко сформулированные, однозначные требования, что даст возможность составить план тестирования и проекты тестов. Если оба коллектива вступают в сотрудничество на ранних стадиях процесса разработки, то велика вероятность того, что им удастся сформулировать необходимые требования уже на ранних этапах временного графика.

Другая причина привлечения к сотрудничеству группы специалистов по тестированию на стадии формулирования и анализа требований к программному продукту продиктована необходимостью проведения статического анализа этих требований. В отчете группы Standish Group по обследованию более чем 350 компаний, опубликованном в 1994 году, сообщается, что только 9% из свыше 8000 проектов создания программных продуктов были выполнены в срок и уложились в финансовую смету [48]. Тридцать один процент всех проектов были аннулированы еще до их завершения. Последующие исследования [49] проводились с целью выявления причин неудачных проектов. Исследование основных факторов, вызвавших перерасход средств на создание продукта или неудачу проекта в целом, показали, что более чем в 50% случаев эти факторы имеют отношение к процессу выработки требований к программному продукту. Основные факторы, имеющие отношение к процессу формулирования требований, перечисляются ниже; там же указано и процентное отношение

проектов, для которых соответствующая проблема стала фактором, повлекшим за собой провал проекта:

- Неполнота требований (13.1%)
- Неучастие пользователя в разработке требований (12.4%)
- Нереалистичные ожидания (9.9%)
- Изменение требований и спецификаций (8.7%)
- Отпала потребность в системе (7.5%).

Одним из результатов этого отчета является вывод о том, что большое число дефектов может быть внесено в продукт уже на стадии формулирования требований. Когда дефекты попадают в требования, возникает их волнообразное распространение на весь процесс разработки, устранение последствий которого требует больших затрат средств и времени. Боем (Boehm) и Папаччио (Papaccio) (которые цитируются в [42]) утверждают, что если затраты на обнаружение и устранение дефекта на стадии формулирования требований составляют $1, то на стадии проектирования эти затраты увеличиваются до $5, на стадии кодирования — $10, на стадии модульного тестирования — $20, а после поставки программного продукта заказчику стоимость работ по устранению того же дефекта достигает $200. Диаграмма, иллюстрирующая затраты на устранение дефекта на разных стадиях разработки, показана на рис. 2.1. По большей части такая эскалация затрат связана с необходимостью повторного выполнения работ на стадии, предшествующей той, на которой этот дефект был обнаружен и устранен.

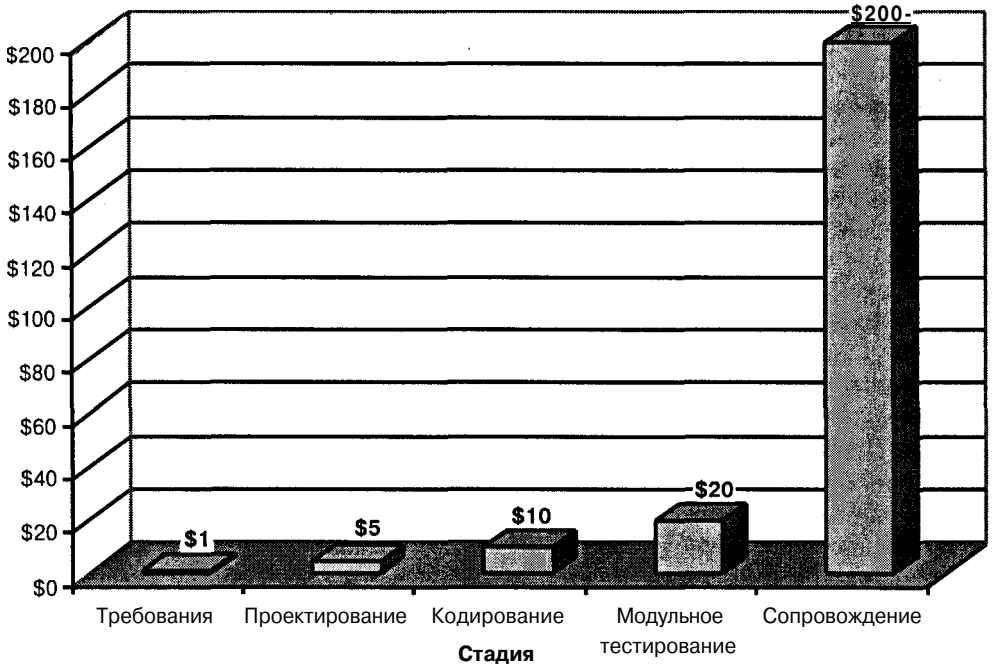



*Рис. 2.1. Стоимость устранения дефекта на стадии разработки. Приводится из [28]*

Опубликованный группой Standish Group анализ данных, который имеет целью определить причины неудачного завершения проектов и стоимость устранения дефектов не на ранних, а на поздних стадиях цикла разработки, недвусмысленно свидетельствует в пользу применения статического тестирования на стадии формулирования требований с тем, чтобы предотвратить проникновение дефектов на более поздние стадии.

Две причины для привлечения тестовой группы для участия на ранней стадии формулирования требований могут быть определены следующим образом:

- Тестовая группа заинтересована в как можно более раннем получении максимально точной спецификации, на основе которой можно составлять план тестирования и проектировать тесты. Это основное условие, без выполнения которого тестирование на ранних стадиях не дает положительного эффекта.
- Тестовая группа должна проводить статическое тестирование спецификаций требований с тем, чтобы предотвратить попадание дефектов на более поздние стадии разработки. Успешное статическое тестирование на стадии формулирования требований позволяет сэкономить время и затраты.

В данной главе мы обсуждаем процесс формулирования требований с точки зрения специалиста по тестированию. Обсуждение начинается с опроса пользователя с целью выявления требований и анализа задач, определения конкретных рабочих продуктов и точек этого процесса, в которых можно применить статическое тестирование. Мы также рассмотрим типы требований, необходимых для обеспечения полноты, и предложим возможные способы тестирования требований. Кроме того, обсуждается возможность использования прототипов программного продукта при анализе требований, равно как и роль тестирования жизненного цикла, основанного на прототипах.

# Процесс формулирования требований

Проекты по созданию программного обеспечения начинают свой жизненный цикл, когда у заказчика возникает необходимость заменить существующую систему или потребность в совершенно новой системе. Заказчиком может быть любой отдел вашей компании, отдельная компания или агентство, которое заключает с вами договор на выполнение работ. В качестве заказчика может также выступать рынок товаров массового производства с активным спросом на готовые программные продукты. Потребности заказчика могут быть представлены набором требований, где под *требованием (requirement)* понимается описание того, что способна выполнять система, либо описание некоторых условий функционирования, которые необходимо обеспечить с тем, чтобы система могла выполнять свои задачи. Требования определяют то, *что* должна выполнять система, но не то, *как* она должна это выполнять. Последнее означает, что в центре внимания требования находится задача заказчика и его деловая активность, но не то, как достигается ее решение.

Рисунок 2.2 служит иллюстрацией процесса формулирования требований к системе программного обеспечения. Мы исследуем этот процесс с точки зрения специалистов по тестированию, которые стремятся получить информацию, поддерживающую планирование тестирования на ранних стадиях разработки, стремятся обнаружить дефекты уже в самих требованиях, чтобы они не инициировали лавину ошибок на более поздних стадиях разработки.


*Рис. 2.2. Процесс формулирования требований*

Первым на рис. 2.2 показан опрос заказчика с целью выявления требований, которые он предъявляет к программному продукту. Выявление требований выполняется в форме вопросов и ответов. С использованием слайдов, макетов и прототипов заказчику предлагаются различные варианты. Сбор пожеланий со стороны заказчика может осуществляться в виде серии интервью или путем использования технологии FAST (Facilitated Application Specification Techniques - технология упрощенной спецификации приложения), представляющей собой специальный тип собеседований с заказчиком, который облегчает выявление его требований.

По мере получения от пользователя, требования должны фиксироваться в виде документа, известного как *документ определения требований (requirements definition document)*. Этот документ оформляется в виде списка требований, сформулированных в результате собеседований с заказчиком. Он представляет собой соглашение между заказчиком и организацией, выполняющей разработки, о том, что должно создаваться. Определение требований представляет собой письменный документ на естественном языке, вполне понятный как заказчику, так и коллективам, которые занимаются разработкой и сопровождением системы.

Достоинство документа определения требований состоит в том, что его легко понять, но в то же время ему не хватает точности и технических деталей, необходимых для проектирования и разработки программного продукта. В силу этого обстоятельства должен быть подготовлен другой документ, известный как *спецификация требований (requirement specification)* или *функциональная спецификация (functional specification)*. Хорошим пособием для первого знакомства с основными понятиями и терминологи-

ей, используемой при подготовке спецификаций требований, могут служить публикации [47], [43] и [42].

Как только документ определения требований и спецификации требований будут готовы, можно приступать к построению *матрицы прослеживаемости требований (requirements traceability matrix).* Назначение этой матрицы состоит в том, чтобы поставить в соответствие каждому требованию тесты, компоненты проекта и программный код. В идеальном случае в этой работе принимают участие коллективы разработчиков и специалистов по тестированию, в результате чего со временем появятся связанные с каждым требованием проектные компоненты, тесты программных модулей, тесты комплексных и приемочных испытаний. При правильном использовании матрица прослеживаемости требований представляет собой инструментальное средство, которое помогает каждому специалисту, принимающему участие в процессе разработки, выполнять работу, непосредственно связанную с потребностями заказчика, и не тратить понапрасну время на решение ненужных задач. С точки зрения группы тестирования этот инструмент может с успехом использоваться для планирования тестирования и проектирования тестов, обеспечивающих хорошее тестовое покрытие.

В нескольких следующих разделах мы проведем более подробное изучение видов деятельности и рабочих продуктов процесса формулировки требований в перспективе тестирования.

## Выявление требований

Обмен информацией между заказчиком и разработчиком в процессе определения требований к программному продукту настолько важен для успеха всего проекта, что для достижения его эффективности затрачиваются значительные усилия и средства. Одним из факторов, способных снизить эффективность такого обмена, является недостаточный объем знаний заказчиком методов разработки программных продуктов. Довольно часто заказчик не настолько разбирается в процессе разработки программного обеспечения, чтобы быть способным выразить свои требования в форме, понятной специалисту по разработке системы.

Процесс определения системы программного обеспечения можно сравнить с проектированием дома по индивидуальному проекту: может случиться так, что вы не настолько хорошо разбираетесь в строительстве, чтобы точно сказать, что вам нужно. Обмен мнениями, по всей видимости, должен включать осмотр уже построенных домов, изучение планировки помещений с целью выбора понравившейся, а также изучение некоторого множества других индивидуальных проектов и чертежей. Что касается систем программного обеспечения, то часто полезно продемонстрировать в работе существующее программное обеспечение, ознакомиться с соответствующими документами или прототипами программного продукта и обсудить рабочую среду, в которой будет использоваться программный продукт. Результаты такой работы с заказчиком вызывают непосредственный интерес и у группы, проводящей тестирование. Специалисты этой группы желают знать, как заказчик намерен использовать программное обеспечение, дабы спроектировать реалистичные тесты для системных и приемочных испытаний.

Один из классов методов, которые могут быть использованы для выявления требований, получил название *технологии FAST* (Facilitated Application Specification Techniques— упрощенной спецификации приложения). Наиболее распространен-

ным подходом в рамках реализации технологии FAST является разработанный компанией IBM *метод JAD* (Joint Application Development — совместная разработка приложения) [43]. Другой метод, *JAR* (Joint Application Requirement — совместная разработка требований к приложению), был предложен Гэри Коббом (Gary Cobb) и описан в главе 8.

Обычно во время сеанса технологии FAST рекомендуется следовать тому или иному набору условий из числа приведенных ниже [43]:

- И заказчики, и разработчики принимают участие в совещании, посвященном вопросам определения требований.
- Установлены правила подготовки и участия в совещании, которым следуют обе стороны. Должна быть установлена атмосфера, содействующая успеху совещания.
- Совещание проводится под управлением посредника. Таким посредником может быть заказчик, разработчик или кто-то посторонний, приемлемый для обеих сторон.
- Для фиксации требований используются такие средства, как лекционные плакаты с рейкой, настенные индикаторные панели или простая ленточная бумага.
- Цель совещания — определить задачи или потребности заказчика, предложить решение, обсудить различные подходы и определить набор требований.

Настоятельно рекомендуется участие в этом совещании специалиста по отладке или даже несколько упростить регламент совещания по технологии FAST с таким расчетом, чтобы в этом совещании могли использоваться элементы статического тестирования. Технология JAR, описанная в главе 8, особо благоприятствует встроенному статическому тестированию, которое в терминологии JAR называется "вспомогательная поддержка".

Одним из видов информации, которую можно извлечь в результате проведения мероприятий по выявлению требований, является соглашение между заказчиком и разработчиком относительно приоритетов требований. Например, каждое требование может быть отнесено в одну из следующих категорий:

- Наиболее важные требования.
- Требования, выполнение которых крайне желательно.
- Требования, выполнение которых желательно, но не обязательно.

Благодаря такому распределению приоритетов упрощается выполнение плана разработок. Например, можно составить план, который ставит выполнение наиболее важных требований в рамках первой версии программного продукта, реализация желательных требований во второй версии и всех остальных — в третьей версии. Аналогично, группа тестирования может начать разработку тестов для требований с наибольшим приоритетом в первую очередь. Такой тип нарастающих поставок или поставок версиями, часто используется в тех случаях, когда решающее значение имеет соблюдение временного графика работ.

Нужно рассмотреть специальный случай, когда от специалистов по тестированию требуется провести испытание программных продуктов, к которым не предъявлено никаких требований, либо случай, когда не все требования выявлены. К сожалению,

упомянутые ситуации встречаются довольно часто; достаточно вспомнить данные группы Standish Group, приведенные в начале главы, в которых первой из причин неудачи проекта или нарушение сроков его разработки была названа неполнота требований. Один из авторов этой книги вспоминает исключительную ситуацию, когда специалисту по тестированию передали компакт-диск с программой и попросили выполнить ее тестирование — но там не было никакой документации, не говоря уже об описании набора требований.

В ситуации, когда спецификация требований отсутствует, у вас, возможно, не остается другого выбора, как только написать собственный документ определения требований. В зависимости от того, сколько времени выделено на завершение тестовых работ, полученное определение требований может оказаться не таким исчерпывающим, как этого бы хотелось, но очень важно, чтобы были определены все ключевые требования. Невозможно тестировать программное обеспечение, если нет более-менее точных определений необходимых функциональных средств. Если вы столкнулись с затруднениями при определении требований, возможно, потребуется провести JAR-сеанс с генеральным разработчиком и, желательно, с представителями группы маркетинга и заказчиком. По меньшей мере, можно будет взять интервью у представителей разработчика и группы маркетинга и сформулировать по возможности полные определения требований, насколько это позволяет отпущенное вам время.

### ВЫЯВЛЕНИЕ ТРЕБОВАНИЙ ПРИ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ РАЗРАБОТКЕ

В условиях объектно-ориентированного похода требования определяются через случаи использования. *Случай использования (use case)* есть описание того, как функция должна выполняться с точки зрения внешнего пользователя или системы. Совокупность всех возможных случаев использования описывает функциональные возможности системы в целом.

Каждый случай использования обычно представлен в формате диаграммы, показывающей используемые объекты плюс текстовое описание (получившее название текста сценария) того, как выполняется соответствующая функция. Обычно проще всего определить системные испытания через случай использования, поскольку такой случай не дает никакого представления о том, как система выполняет соответствующую функцию, зато четко определяет, какие функциональные возможности должны быть реализованы. Следовательно, в основу проекта системных испытаний непосредственно могут быть положены случаи использования, при этом для каждого случая использования разрабатывается, по меньшей мере, один тест.

Информация о тестировании случаев использования изложена в главе 10.

*Врезка 2.1*

## Определение требований

Документ определения требований содержит все требования, предъявляемые к системе и сформулированные на естественном языке, благодаря чему они становятся понятными и заказчику, и тем, кто принимает участие в разработке проекта. В перспективе тестирования мы заинтересованы в получении из этого документа информации, достаточной для того, чтобы иметь возможность приступить к планированию и разработке тестов. Для наших целей определения требований должны обладать следующими свойствами:

- Каждое требование должно быть снабжено уникальным идентификатором, чтобы можно было однозначно ссылаться на него при планировании тестового покрытия, при проектировании тестовых случаев и в отчетах по результатам тестирования.
- Требования должны быть представлены с точки зрения пользователя системы. Системные и приемочные испытания должны быть спроектированы на основе определений требований, следовательно, эти определения должны быть сформулированы в перспективе системного уровня. Этот принцип препятствует появлению требований, затрагивающих внутренние свойства системы и требующих детальных знаний программного кода для успешного тестирования. Такие требования должны возникать на поздних стадиях разработки и охватываться модульным тестированием и проверкой взаимодействия и функционирования компонентов системы.
- Должны быть включены как *функциональные (functional),* так и *нефункциональные (nonfunctional)* требования. Функциональные требования суть требования, описывающие услуги и функции, которые должна выполнять разрабатываемая система. Нефункциональные требования описывают ограничения, накладываемые на работу системы, например, количество одновременно работающих пользователей, и стандарты, которым должна соответствовать система.
- Документ определения требований должен находится под управлением конфигурациями. Это, по меньшей мере, означает, что данный документ подпадает под управление версиями и что все версии документа должны быть помещены в безопасное хранилище, подобное, например, каталогу, содержимое которого обычно дублируется. Если требования подвергаются изменениям, мы должны иметь возможность проследить, чтобы соответствующие изменения были внесены в тестовые случаи системных и приемочных испытаний.

Оглавление одного из возможных вариантов документа, содержащего системные требования, представлено на рис. 2.3. Это оглавление соответствует стандарту IEEE Standard 830: *The IEEE Guide to Software Requirements Specifications* [23] — Руководящие принципы IEEE по составлению спецификации требований к программному обеспечению. Пример документа определения требований, соответствующего рис. 2.3, приводится в третьей части книги. Этот пример может принести определенную пользу, если вы окажетесь в ситуации, когда необходимо будет составить собственный документ определения требований. Кроме того, он может оказаться полезным при проведении статического тестирования различных документов, основанных на определении требований.

**Документ определения требований (Requirements Definition Document)**
**Оглавление (Table of content)**



***Рис. 2.3. Пример оглавления документа определения требований к программному продукту. Составлен в соответствии со стандартом IEEE Standard 830: The* IEEE Guide to Software Requirements Specifications [23]. *Все права защищены.***

Документ определения требований, показанный на рис. 2.3, включает общее описание, которое показывает, какими достоинствами обладает данный продукт перед другими продуктами этого типа, и описание его функциональных возможностей. В нем дано краткое описание основных свойств продукта и указано, какой квалификацией должен обладать пользователь, чтобы работать с ним. В то же время документ не содержит подробного описания функциональных средств. Раздел специальных требований содержит определения функциональных требований, требований к производительности, к интерфейсам и ряд других требований. Рассматриваемый документ должен содержать определения сокращенных обозначений (акронимов) и технической терминологии, используемой для описания продукта.

Как только документ определения требований будет написан, он должен быть подвергнут статическому тестированию с целью проверки требований на полноту, непротиворечивость, осуществимость, контролепригодность, однозначность и релевантность.

**Типы требований.** Выше мы определили требования как описание того, что способна сделать система, либо как условия эксплуатации, которые необходимо обеспечить для того, чтобы система была способна выполнять возложенные на нее задачи. Это подразумевает использование широкого диапазона различных сведений,

поэтому полезно разбить требования на категории. Пример набора таких категорий представлен следующим списком:

- **Функциональные средства.** Этот набор требований определяет, какие функции должен выполнять данный программный продукт на системном или пользовательском уровне. Для ясности может быть указано, чего *ие должен* делать данный продукт.
- **Интерфейсы.** Эта категория требований описывает входы, получаемые из внешних систем, и выходы, направляемые во внешние системы. Накладываются ли на эти интерфейсы какие-то ограничения, связанные с форматами данных и носителями информации?
- **Данные.** Эти требования описывают входные и выходные данные системы. Какой при этом используется формат? Какие данные нужно сохранять? Какой объем данных поступает в систему и из системы, и с какой скоростью передачи? С какой точностью должны выполняться вычисления?
- **Производительность.** Требования этой категории описывают проблемы масштабирования и синхронизации, например, сколько пользователей одновременно должна обслуживать система, сколько транзакций должна выполнять система в единицу времени, как долго пользователь должен ждать ответа на свой запрос. Следует соблюдать особую осторожность при выражении этих требований в числовых значениях, поскольку в дальнейшем их придется подтверждать.
- **Пользователи и человеческий фактор.** Эта категория требований предъявляется к тем, кто будет работать с системой в смысле их квалификации и опыта. Они также определяют необходимый уровень удобства и простоты использования программного продукта, например, сколько отдельных действий требуется для выполнения рутинной операции в системе? Насколько отчетливо выражена обратная связь после каждого действия?
- **Физическая среда.** Эта категория требований определяет, где должно функционировать оборудование. Установлено ли оборудование более чем в одном месте? Имеют ли место ограничения по температуре, влажности или другие ограничения, связанные с окружающей средой?
- **Безопасность.** Эта категория требований описывает, как осуществляется доступ к системе и как осуществляется управление ее данными. Данная категория является подходящим местом для описания, как должны дублироваться системные данные. Как часто следует выполнять дублирование? На каких носителях сохраняются дублированные данные?
- **Документация.** Эта категория требований связана с документацией и тем, должна ли она быть представлена в печатном виде или выдается в интерактивном режиме. Кроме того, здесь же определяется категория лиц, для которых эта документация предназначена.
- **Устранение неисправностей.** Эта категория требований описывает, как система реагирует на возникновение неисправностей. Будет ли система обнаруживать неисправность и выдавать аварийные сигналы? Какой должна быть

средняя длительность промежутка времени между двумя неисправностями? Каким должно быть максимальное время простоя?

- **Сопровождение.** Эти требования определяют, как производится устранение проблем, обнаруженных в системе, как усовершенствованные версии системы поставляются заказчику и как пользователи должны переходить на усовершенствованную версию системы.
- **План поставки версий.** Если требованиям назначены приоритеты, возможно, необходимо будет определить временной график поставки версий, который показывает, реализации каких требований уделяется внимание в конкретной версии.

Выполняя статическое тестирование на документах с формулировками требований, возможно, возникнет желание воспользоваться приведенным выше списком с целью обеспечения полноты тестирования. Если вы ведете предысторию статического тестирования сразу нескольких проектов, этот список можно расширить таким образом, чтобы он соответствовал номенклатуре программных продуктов, создаваемых вашей компанией. Если обстоятельства заставляют вас проводить тестирование в условиях неполного набора требований, на основе предложенного списка, скорее всего, придется определить, какие тесты нужны помимо тех, что образуют покрытие задокументированных требований.

**Примеры требований.** Пример, содержащий несколько функциональных требований, приводится на рис. 2.4. Этот пример относится к гипотетическому продукту, который мы будем "разрабатывать" на протяжении практически всей книги. Указанный демонстрационный программный продукт, который мы назовем комплектом TMT (Test Management Toolkit — инструментальные средства управления тестами), представляет собой приложение, предназначенное для тестовых групп, которым необходимы автоматизированные инструментальные средства для поддержки планирования тестирования, разработки тестовых случаев и выполнения тестов. Определение требований к комплекту TMT можно найти в третьей части книги.

Требования, представленные на рис. 2.4, касаются некоторых ключевых свойств программного продукта.

## Спецификация требований

Спецификация требований описывает то же, что и документ определения требований, однако спецификация предназначена для системных разработчиков и представлена на языке или в обозначениях, ориентированных на разработчиков. Спецификацию требований часто называют системной функциональной спецификацией, несмотря на то что ее область действия распространяется за пределы на функциональных требований системы.

Спецификация требований, или функциональная спецификация, — это далеко не примитивный перевод определений требований на технический язык. Спецификация может проводить дальнейшее разбиение требований по категориям, а также добавлять подробности за счет формулирования некоторого набора производных требований. Например, в исходном определении требований может быть указано, что

система должна работать в заданном диапазоне температур, в то время как в спецификации могут быть определены различные режимы эксплуатации системы в различных температурных диапазонах. В какой-то мере определение требований представляет собой объявление того, что заказчику необходимо и что он хотел бы получить, а спецификация есть ответное предложение группы специалистов, содержащее описание того, что будущая система будет способна делать.

**Требования к инструментальным средствам управления тестами**

2.2.1 Приложение должно предоставить средства создания, модификации, просмотра, хранения и поиска документов, имеющих отношение к планированию тестирования.

2.2.2 Приложение должно предоставить средства создания, модификации, просмотра, хранения и поиска списка тестов, подлежащих выполнению. В этом документе список этих средств будет называться "списком прогона".

2.2.3 Приложение должно предоставить средства создания, модификации, просмотра, хранения и поиска индивидуальных тестов, которые могут состоять из таких компонентов как процедура установки, список оборудования, методика испытаний, контрольные данные и тестовые процедуры очистки.

2.2.4 Приложение должно предоставить такие средства для выполнения тестов из списка прогона, чтобы для каждого теста можно было создавать результаты тестирования и регистрационные журналы.

2.2.5 Приложение должно предоставить средства создания, модификации, просмотра, сортировки, хранения и поиска сообщений о дефектах.

2.2.6 Приложение должно предоставить средства для генерации отчетов, в которых подводятся итоги состояния тестирования, результатов тестирования и сообщений о дефектах.

***Рис. 2.4. Пример определения требований***

Составители спецификации требований могут использовать естественный язык либо выбрать язык или обозначения из широкого набора языков, разработанных специально для написания спецификации требований. Поскольку термины в естественном языке могут интерпретироваться различными способами, их применение может усилить взаимное непонимание между заказчиком и разработчиком, если не предпринять специальных мер по уточнению значений отдельных слов, таких как, например, "производительность", "практичность", "надежных" и т.п. Особенности привлечения для формулирования требований языков, отличных от естественных, описано в [42].

Готовая формальная спецификация требований должна быть подвергнута статическому тестированию с целью проверки, что каждое требование, зафиксированное в документе определения требований, отражено в спецификации требований, а каждое требование одного документа отображается или отслеживается из другого документа. Окончательная редакция спецификации должна пройти статическое тестирование с тем, чтобы убедиться, что требования спецификации обладают свойствами полноты, непротиворечивости, осуществимости, контролепригодности, однозначности и релевантности.

## Матрица прослеживаемости требований

Назначение матрицы прослеживаемости требований заключается в отображении каждого требования на проектные компоненты, программные коды и тестовые случаи. Пример матрицы прослеживаемости требований показан на рис. 2.5. В этом примере каждое требование, фигурирующее в документе определения требований, отображается на одно или большее количество требований из спецификации, на проектные компоненты и программные коды, а также на тестовые случаи модульного тестирования, проверки взаимодействия и функционирования компонентов программного продукта, системных и приемочных испытаний. Для нумерации требований используется "десятичная нотация Дьюи". Эта нотация позволяет отобразить исходные требования из документа определения требований на производные требования и различные виды тестов в отношении "один-ко-многим" (например, одно требование соотносится с некоторым множеством системных тестов). Даже если проектная информация и данные о программных кодах не включены в эту матрицу, полезно поддерживать ее упрощенный вариант, который соотносит каждое требование с соответствующими системными и приемочными тестами.

Эти идентификаторы отображают требования из документа определения требований на требования из спецификации требований.

Эти идентификаторы отображают все тесты обратно на требования.

| Идентификатор требования в документе определения требований | Идентификатор требования в спецификации требований | Идентификатор проектного компонента | Идентификатор программного компонента | Идентификатор тестового случая на этапе модульного тестирования | Идентификатор тестового случая на этапе проверки взаимодействия компонентов продукта | Идентификатор тестового случая на этапе системных испытаний | Идентификатор тестового случая на этапе приемочных испытаний |
|---|---|---|---|---|---|---|---|
| **RD2** 2 4 | RS2 2 4 1 | D2241 | CC2 2 4 1 | UT2 2 4 1 | IT2 2 4 | ST2 2 4 | AT2 2 4 |
| RD2 2 4 | RS2 2 4 2 | D2 2 42 | CC2 2 4 2 | UT2 2 4 2 | IT2 2 4 | ST2 2 4 | AT2 2 4 |
| **RD2** 2 4 | RS2 2 4.3 | D2 2 4 3 | CC2 2 4 3 | UT2 2 4 3 | IT2 2 4 | ST2 2 4 | AT2 2 4 |
| RD2 2 4 | RS2 2 4 4 | D2 2 44 | CC2 2 4 4 | UT2 2 4 4 | IT2 2 4 | ST2 2 4 | AT2 2 4 |

Эти идентификаторы прослеживают происхождение всех проектных и программных компонентов до соответствующих требований.

***Рис. 2.5.*** *Примерный* ***формат матрицы отслеживания требований***

# Тестирование требований

Завершающим действием процесса формулирования требований является статическое тестирование требований. Одна из целей статического тестирования предусматривают проверку требований на полноту, непротиворечивость, осуществимость, а также на возможность тестирования (статического или динамического) в процессе реализации. Дополнительная цель такого статического тестирования заключается в проверке того, что требования, представленные в окончательном виде, соответству-

ют пожеланиям заказчика, выраженных им на стадии выявления требований. Правильно выполненное статическое тестирование оборачивается серьезным выигрышем в смысле экономии средств и времени. Напомним, что согласно данным группы Standish, рассматриваемым в начале главы, проблемы, возникающие на этапе формулирования требований, порождают более 50% проектных ошибок и перерасхода ресурсов. Статическое тестирование требований представляет собой наиболее эффективный способ обнаружения этих дефектов еще до того, как они окажут неблагоприятное влияние на планы выполнения работ и сметные расходы.

Существует множество способов выполнения статического тестирования, однако тремя наиболее распространенными являются инспекции, сквозной контроль и экспертные оценки. Иногда названия этих методов звучат по-другому; инспекции известны как технический анализ, а экспертные оценки нередко называют "партнерским контролем". Характерные особенности каждого из этих типов статического тестирования приводятся в таблице 2.1. Более подробную информацию о методах статического тестирования можно найти во врезке 2.2 и в главе 9.

**Таблица 2.1. Характерные особенности избранных методов статического тестирования [28]**

| | Инспекции | Сквозной контроль | Экспертные оценки |
|---|---|---|---|
| **Презентатор** | Любой, но не автор | Любой | Нет |
| **Число участников** | 1-6 человек | 5 или более | 1 или два |
| **Подготовка** | Да | Только презентатор | Нет |
| **Сбор данных** | Да | Не обязательно | Нет |
| **Отчет по результатам** | Да | Не обязательно | Словесный комментарии или комментарий по электронной почте |
| **Достоинства** | Максимальная эффективность | Большее число участников | Минимальные затраты |
| **Недостатки** | Наибольшие затраты | Обнаруживает меньше дефектов, чем другие способы | Обнаруживает минимальное число дефектов по сравнению с другими способами |

Из таблицы 2.1 следует, что в то время, как затраты на проведение инспекций больше, чем на реализацию других методов, они обеспечивают наивысшую эффективность обнаружения дефектов в проверяемом рабочем продукте. Поскольку очень важно найти как можно больше дефектов на стадии формулирования требований, мы настоятельно рекомендуем инспекции как основной метод анализа требований.

**СТАТИЧЕСКИЕ МЕТОДЫ ТЕСТИРОВАНИЯ**

Наиболее распространенными методами статического тестирования являются инспекции, сквозной контроль и экспертные оценки. Основные характеристики каждого метода приводятся ниже в данном разделе.

**Инспекции**

Основной организационной формой инспекции является совещание, на котором рабочий продукт анализируется с целью обнаружения дефектов. Каждый участник специально готовится к совещанию, а само совещание проводится в соответствии со специальным набором правил. Обнаруженные дефекты документируются, итоги совещания публикуются. Практика показала высокую эффективность таких сообщений с точки зрения обнаружения дефектов. Данные, опубликованные Бремом [28], показывают, что если на инспекции приходится 20% трудозатрат на программирование, то из инспектируемого программного кода будет изъято примерно 80% ошибок на уровне программных модулей. Первоначально рекомендации по проведению инспекций были сформулированы Фейганом [16] из компании IBM, причем с течением времени другие организации приняли их в качестве стандартов или рекомендаций для практической деятельности.

**Сквозной контроль**

Сквозной контроль представляет собой менее формальное мероприятие, чем инспекции, в том смысле, что ни от одного из его исполнителей не требуется специальной подготовки, за исключением разве что презентатора. Кроме того, никаких итоговых отчетов при этом не требуется. Поскольку сквозной контроль формализован в меньшей степени, нежели инспекции, то он может охватывать больше материала. В то же время он не обладает такой эффективностью обнаружения и документирования дефектов, как инспекции.

**Экспертные оценки**

Экспертные оценки требуют немного больше, чем просто передача рабочего продукта коллеге по работе и выслушивание его мнения по этому поводу. Экспертная оценка может быть выражена словесно либо передана по электронной почте. Формальные процедуры экспертных оценок отсутствуют, эффективность поиска и документирования дефектов, свойственная этому методу, меньше, чем эффективность двух других методов.

*Врезка 2.2.*

## Критерии, используемые при тестировании требований

Существует шесть базовых критериев, которые должны использоваться во время статического тестирования спецификаций требований. Критерии требуют, чтобы каждое требование отвечало принципам полноты, однозначности, непротиворечивости, прослеживаемости, осуществимости и контролепригодности.

**Полнота.** Набор требований считается полным, если все его составные части представлены и каждая такая часть выполнена в полном объеме. При тестировании набора требований на полноту необходимо обратить особое внимание на следующие моменты:

- Требования не должны содержать выражений типа "подлежит определению", "и так далее", "и прочие" и им подобных.
- Требования не должны ссылаться на несуществующую справочную информацию, такую как, например, несуществующая спецификация.
- Требование не должно ссылаться на функциональные средства, которые еще не определены.

**Однозначность.** Каждое требование должно быть точно и ясно сформулировано; оно должно допускать единственное толкование. Требование должно быть удобочитаемым и понятным. Если требование отличается особой сложностью, для облегчения понимания может быть привлечен вспомогательный материал, такой как, диаграммы или таблицы. Если для пущей убедительности используются выражения наподобие "это очевидно" или "само собой разумеется", то вполне возможно, что автор пытается отвлечь ваше внимание от того или иного двусмысленного утверждения.

**Непротиворечивость.** Требования не должны противоречить друг другу или действующим стандартам. Если требования конфликтуют друг с другом, то нужно вводить приоритеты с целью разрешения таких конфликтов. Умение обнаруживать дефекты, обусловленные противоречиями требований, предполагает хорошее знание всего документа, содержащего требования, и знакомство с существующими стандартами или другими внешними техническими условиями.

**Прослеживаемость.** Каждое требование должно иметь уникальный идентификатор, который позволяет прослеживать его разработку на протяжении всего жизненного цикла. В рабочих продуктах, которые появляются на более поздних этапах жизненного цикла, таких как план тестирования, каждая ссылка на свойство системы должна прослеживаться до определения и спецификации требований. Матрица прослеживаемости требований, обсуждавшаяся ранее в этой главе, является отличным инструментальным средством для решения упомянутой задачи.

**Осуществимость.** Каждое требование должно ставить перед системой задачу обеспечить такие средства, которые целесообразно разрабатывать и поддерживать. Если заказчик предъявляет к системе нереальные требования в смысле затрат времени и средств на разработку тех или иных функций либо же требует разработки функций, которые окажутся ненадежными и опасными в эксплуатации, необходимо определить риски и принять соответствующие меры. По сути дела, разрабатываемая система должна быть экономически осуществимой, надежной, удобной в эксплуатации и сопровождении.

**Контролепригодность.** Мы должны уметь разрабатывать экономически обоснованные и удобные для использования тесты для каждого требования с целью продемонстрировать, что тестируемый программный продукт обладает необходимыми функциональными возможностями, рабочими характеристиками и соответствует действующим стандартам. Это означает, что каждое требование должно быть измеряемым или поддаваться количественному определению и что тестирование должно выполняться в приемлемых условиях.

## Использование прототипов

В качестве дополнения к методам статического тестирования для проверки и подтверждения требований с большой пользой могут быть использованы прототипы программного продукта. *Прототип (prototype),* будь то макет на бумаге или модель программного продукта, дает вам возможность предложить заказчику варианты и получить обратную реакцию, которая позволяет сделать формулировки требований более точными. Прототипы часто позволяют выявить дефекты в полноте, непротиворечи-

вости или в осуществимости спецификаций требований, и в силу этого обстоятельства могут служить хорошим дополнением статического тестирования.

Существуют два подхода к использованию прототипов. В рамках одного из них производится построение одноразового прототипа (throwaway prototype), который используется исключительно для определения требований; прототип не поставляется заказчику в виде программного продукта. Второй подход предусматривает создание эволюционного прототипа (evolutionary prototype), который применяется на начальной стадии процесса для выявления и анализа требований, и в то же время подвергается многократным усовершенствованиям, постепенно превращаясь в продукт, который уже можно поставлять заказчику. Тестирование эволюционного прототипа программного продукта рассматривается в следующем разделе.

Один из способов встраивания прототипа в жизненный цикл разработки показан на рис. 2.6 [42]. В рамках этого подхода прототипы могут быть построены на стадии формулирования требований и на стадии проектирования цикла разработки. Прототипы используются во время анализа требований для уточнения и тестирования требований. На стадиях системного проектирования и проектирования программ разработчики могут применять прототипы для оценки альтернативных стратегий проектирования. Программный код, разработанный для прототипа, в такой модели разработки может как использоваться, так и отбрасываться за ненадобностью.

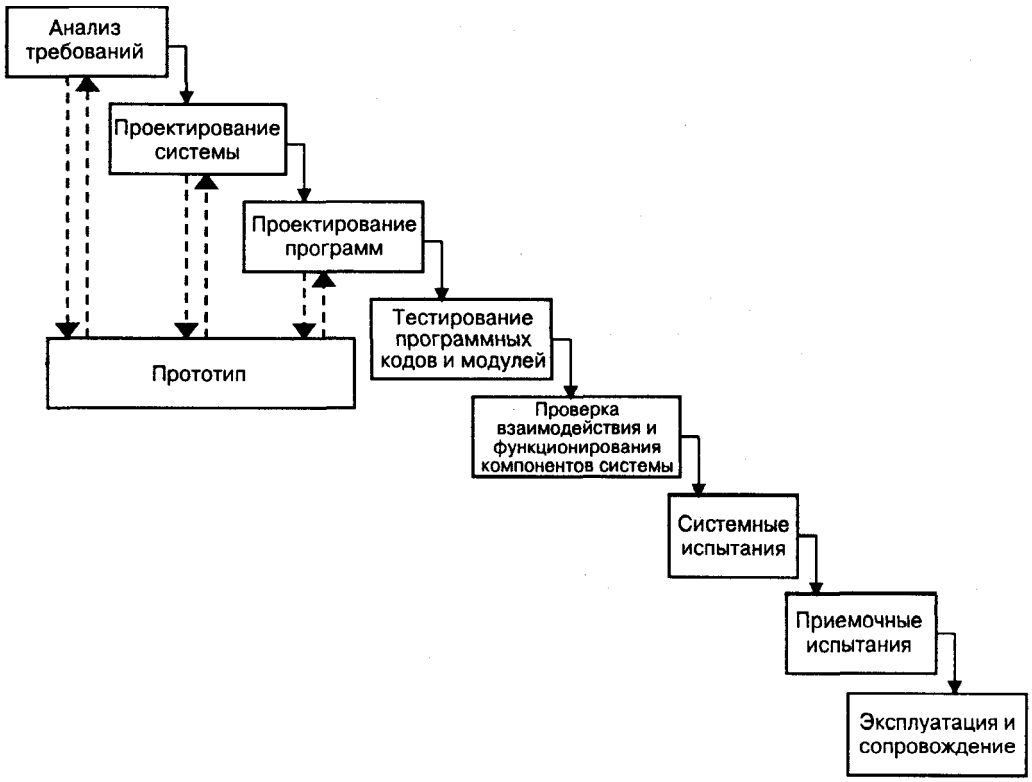



*Рис. 2.6. Прототипу добавленные в каскадную модель жизненного цикла. Взято из [42]; измерения внесены с разрешения издательства Prentice* ***Hall***

Чтобы получить преимущество в начале тестовых работ, тестовая группа может воспользоваться прототипами, разработанными на стадии анализа требований. Можно разработать предварительные испытания и выполнить их на прототипе с целью подтверждения правильности основных требований. Эти тесты будут совершенствоваться на более поздних стадиях в процессе формирования ключевого набора тестов для этапов системного тестирования и приемочных испытаний. Подобно тому, как прототип помогает сделать требования "более реалистичными" с точки зрения заказчика и разработчика, этот прототип позволяет тестовой группе "увидеть", как следует определять набор тестов для системы. Если прототипом является модель на бумаге или набор эскизов, то ему придется пройти долгий путь, пока он начнет приносить тестовой группе реальную помощь в понимании того, как получить полезный набор тестов.

Применение прототипов не только помогает получить тестовой группе преимущества на старте, оно поддерживает статическое или динамическое тестирование на стадии формулирования требований. Прототипы могут использоваться в качестве вспомогательного средства для определения полноты, непротиворечивости, осуществимости и релевантности требований.

Прототипы используют в моделях жизненных циклов, отличных от каскадных. По существу, в случае эволюционного прототипа оно может стать основой процесса разработки. Одной из наиболее сложных моделей жизненного цикла является спиральная модель, разработанная Боемом [42]. Спиральная модель реализует подход, учитывающий риски, она разбивает проект на некоторую последовательность "мини-проектов", каждый из которых решает ту или иную крупную задачу. После того как все основные риски будут учтены, модель завершается обычной каскадной разработкой конечного программного продукта.

## Тестирование в рамках жизненного цикла эволюционного прототипирования

Создание эволюционных прототипов представляет собой метод поэтапной разработки системы, при этом каждый этап определяется ответной реакцией заказчика и результатами тестирования. Прототипы особенно полезны, когда вы испытываете затруднения при определении основных требований на начальной стадии проекта и в то же время имеете возможность многократно контактировать с заказчиком, чтобы понять, что ему нужно. На каждой итерации цикла разработки приходится прибегать к помощи как статического, так и динамического тестирования. Пример жизненного цикла показан на рис. 2.7.

В эволюционной модели разработка начинается с определения исходного прототипа. Если разработка ведется по спиральной модели, то в центре внимания исходной модели может находиться оценка основных рисков проекта. В других случаях исходным прототипом может быть макет пользовательского интерфейса, в условиях которого пользователи образуют обратную связь в рамках функциональных возможностей и удобства и простоты обслуживания системы. Статическое тестирование следует применять в отношении разработки исходного прототипа в виде экспертиз или совместного с заказчиком рабочего совещания, например, JAR-сеанса.

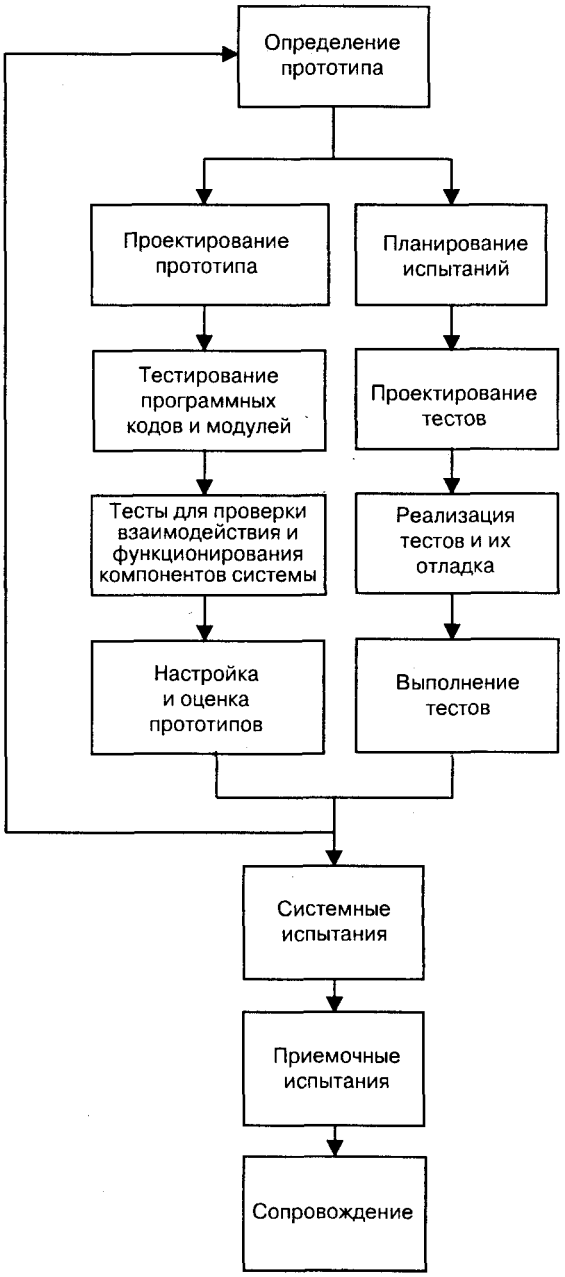



*Рис. 2.7. Тестирование в рамках жизненного цикла эволюционного прототипирования*

Как только определение прототипа завершается, группа разработчиков выполняет проектирование, кодирование и тестирование прототипа, в то время как группа тестирования параллельно разрабатывает планы тестирования, создает тесты и выполняет их прогон. Эта модель требует тесного взаимодействия между коллективом разработчиков и группой тестирования с тем, чтобы каждый прототип подвергся тестированию в достаточных объемах перед его демонстрацией пользователю. Если функциональные возможности системы возрастают с каждой итерацией, группа тестирования должна иметь возможность выполнить регрессионное тестирование на каждом шаге цикла разработки. Другими словами, необходимо прогонять специальные тесты, чтобы убедиться, что старые функциональные средства не разрушены или не деградировали в результате добавления новых функциональных возможностей. Автоматизация может оказаться исключительно полезной для регрессионного тестирования. Если количество тестов, выполняемых вручную, возрастает с каждым циклом разработки, трудозатраты на тестирования существенно возрастают и- требуют все больше и больше времени.

По завершении достаточно большого числа циклов разработки перед передачей конечного программного продукта заказчику потребуется провести завершающие системные и приемочные испытания продукта. Из примера 2.7 нетрудно видеть, что все основные функции динамического тестирования (планирование тестирования, проектирования, реализации и выполнение) выполняются в рамках жизненного цикла эволюционного прототипирования, но они выполняются в итеративном режиме, что обусловливает необходимость регрессионного тестирования.

# Что дальше

В данной главе были проведены исследования стадии формулирования требований процесса разработки с точки зрения специалиста по тестированию систем. Мы обнаружили два основных фактора, обусловливающих участие группы тестирования на стадии формулирования требований:

- Необходимость получения спецификаций, служащих основой планов тестирования и проектирования тестов.
- Необходимость статического тестирования спецификаций требований с целью предотвращения перетекания дефектов на более поздние стадии разработки.

### ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА УПРАВЛЕНИЯ ТРЕБОВАНИЯМИ

В продаже можно найти несколько программных инструментальных средств, которые можно использовать на этапе формулирования требований. Программные инструментальные средства особенно полезны в плане выявления требований и обеспечения прослеживаемости требований на протяжении процесса разработки. Примерами средств управления требованиями являются DOORS и Requisite Pro. Инструментарий DOORS -это инструментальное средство, разработанное компанией Quality System and Softwaie (QSS) Ltd. Еще одно средство — это модуль Requisite Pro, который продается компанией Rational Software. Компания Atlantic Systems Guild, Inc. занимается независимыми исследованиями инструментальных средств управления требованиями, результаты которых публикуются на Web-сайте *http://www.systemguild.com*.

***Врезка 2 3***

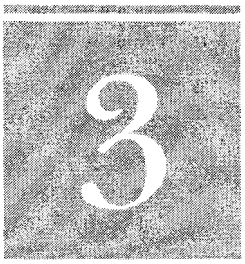
Мы выявили три вида рабочих продуктов, которые создаются на стадии формулирования требований:

- Документ определения требований, который представляет собой изложение требований заказчика к программному продукту, сформулированное на естественном языке.
- Спецификация требований, которая еще называется функциональной спецификацией, представляющая собой технические требования, выраженные в нотации, которая может использоваться для разработки программных продуктов.
- Документ, отслеживающий требования, который может применяться для отображения теста на требование, послужившее причиной появления этого теста. В результате обеспечивается возможность тестирования всех требований.

Мы описали содержимое каждого из этих документов и выдвинули некоторые идеи относительно того, как их проверять. Более подробную информацию о применении статического тестирования на стадии формулирования требований, включая описание методики проведения JAR-сеансов, можно найти в главе 8. Наконец, мы обсудили роль прототипирования на стадии формулировки требований и рассмотрели, как выполняется тестирование в рамках жизненного цикла эволюционного прототипа.

В следующей главе мы более подробно рассмотрим, что происходит после выявления всех требований к программному продукту. В центр внимания выдвигаются планирование тестирования и оценка трудозатрат.

# Планирование испытаний



**Темы, рассматриваемые в главе:**

- Стратегия тестирования
- Определение стратегии тестирования
- Оценка трудозатрат на тестирование
- Подготовка и пересмотр документов, содержащих планы проведения испытаний
- Что дальше

Как отмечалось с самого начала книги, основной принцип быстрого тестирования предусматривает максимально тесную связь с разработкой на протяжении всего жизненного цикла программного продукта. Во время выявления и анализа требований подобное интегрирование достигается за счет использования технологии статического тестирования. Последняя предусматривает пересмотр требований с целью обнаружения дефектов, а также участие специалистов по тестированию в процессе формулирования требований с целью более раннего получения спецификаций, что, в свою очередь, позволяет задействовать требования во время планирования испытаний. Принцип интегрирования разработки и тестовых действий должен соблюдаться и в процессе планирования и определения трудозатрат. Тестирование программного обеспечения представляет собой дорогостоящую, ресурсоемкую работу, на выполнение которой расходуется до половины сметной стоимости проекта. Высокая эффективность планирования испытаний и правильная оценка трудозатрат в значительной мере содействует успеху всего процесса тестирования, в то время как неудачи на этой стадии могут привести к превышению сметной стоимости проекта и нарушению графика работ.

Диаграмма видов деятельности, связанных с планированием испытаний, показана на рис. 3.1. Входными для процесса планирования будут документы, которые содержат требования, описанные в главе 2. Как упоминалось в главе 2, если документы с формулировками требований отсутствуют, придется самостоятельно составить, по меньшей мере, сокращенный вариант таких документов. Невозможно провести необходимое тестирование программного продукта, не имея представлений о функциональных возможностях продукта и связанных с ними ожиданий заказчика.


*Рис. 3.1. Виды деятельности, осуществляемые **при** составлении плана **испытаний***

Выходным результатом действий исполнителей, осуществляющих планирование тестирования, является документ или набор документов, который должен быть проверен тестовой группой, группой разработчиков и персоналом, осуществляющим управление разработкой и сопровождением программ. В плане проведения испытаний указаны ресурсы, необходимые для тестирования программного продукта, определено, что подлежит тестированию, как должно проводиться тестирование и какие выходы или выходные результаты будут получены по итогам тестирования. Содержимое и формат плана испытаний обсуждаются далее в этой главе.

Процесс планирования испытаний образуется из следующих основных видов деятельности:

- Определение стратегии тестирования
- Определение состава и структуры испытательной системы (аппаратных и программных средств)
- Оценка трудозатрат (ресурсы и график работ)
- Оценка рисков невыполнения графика работ и подготовка плана мероприятий по смягчению последствий от овеществления этих рисков.
- Подготовка и пересмотр (утверждение) документов с планом проведения ис пытаний.

Каждый из указанных выше видов деятельности подробно обсуждается в данной главе. Технология оценки трудозатрат исследуется в главе 12. Пример плана прове

дения испытаний приложения TMT приводится в третьей части книги. Несмотря на то что на рис. 3.1 показана линейная последовательность основных видов деятельности, зачастую они выполняются одновременно или по итерационной схеме. И хотя порядок выполнения видов деятельности может меняться, важно, чтобы все они были частью процесса планирования испытаний.

Планирование тестирования можно сравнить с луковицей, т.е. это многоуровневый процесс. Планирование начинается с определения стратегии тестирования на концептуальном уровне, после чего добавляются уровни дальнейшей детализации, которые описывают архитектуру тестирования, условия испытаний, а также тестовые случаи до тех пор, пока не будет составлен план проведения испытаний в его окончательном виде. После того, как план проведения испытаний будет оформлен в виде документа и утвержден, процесс планирования не прекращается, поскольку возможны изменения требований, изменения в графике работ и другие виды изменений, которые влекут за собой коррекцию плана проведения испытаний. План проведения испытаний должен поддерживаться на всем протяжении процесса разработки программного продукта как динамически развивающийся документ. Он должен храниться в безопасном хранилище и поддаваться управлению версиями.

## Стратегия тестирования

Первое действие в планировании испытаний предусматривает разработку стратегии тестирования на высоком уровне. В общем случае стратегия тестирования должна определять объемы тестовых работ, типы методик тестирования, которые должны применяться для обнаружения дефектов, процедуры, уведомляющие об обнаружении и устраняющие дефекты, критерии входа и выхода из испытаний, которые управляют различными видами тестирования. Реализуя принцип тесного интегрирования разработки и тестирования с целью оптимизации графика разработки, стратегия тестирования должна отображать различные виды тестовой деятельности на жизненный цикл разработки. При формулировании общей стратегии должно быть предусмотрено как статическое, так и динамическое тестирование.

Если для поддержки различных видов тестовой деятельности используется автоматизация, стратегия автоматизации должна рассматриваться как составная часть общей стратегии тестирования. Автоматизация требует выполнения независимых параллельных работ, которые должны тщательно планироваться и выполняться только в тех случаях, когда это не приводит к снижению эффективности.

Мы предлагаем следующий подход к формулированию стратегии тестирования:

1. Определить объемы тестовых работ путем анализа документов, содержащих требования к программному продукту (технические условия), дабы выяснить, что нужно тестировать. Рассмотреть виды тестирования, которые не следуют непосредственно из документов с требованиями, такие как тестирование возможности установки и наращивания возможностей программного продукта, удобство и простота обслуживания продукта, а также способности к взаимодействию с другими видами аппаратных средств из среды заказчика.
2. Определить подход к тестированию за счет выбора статических и динамических тестов, связанных с каждой стадией разработки. Здесь потребуется

включить описания всех рабочих продуктов, которые должна подготовить тестовая группа.

3. Определить критерии входа и выхода для каждой стадии тестирования, равно как и все точки контроля качества, для чего потребуется участие специалистов по тестированию.
4. Определить стратегию автоматизации в случае, если планируется использование автоматизации какого-либо вида тестовой деятельности. Автоматизация требует проведения независимых параллельных работ, которые должны тщательно планироваться и выполняться только в тех случаях, когда это не приводит к снижению эффективности.

## Определение объемов тестовых работ

При определении объемов тестовых работ важно рассмотреть вопрос, какая часть программного продукта должна подвергаться тестированию. В области тестирования программного обеспечения давно установлен тот факт, что программу, реализующую большую часть функциональных возможностей, невозможно протестировать полностью. Этот факт исследуется во врезке 3.1 "Можно ли выполнить исчерпывающее тестирование программы?" Ответом, который мы получаем на основе этой врезки, будет "Нет!". Конечно, можно выполнить достаточно исчерпывающее тестирование старой доброй программы, выдающей фразу "hello, world" (привет, мир!), которая содержит всего лишь несколько строк, не имеет условных переходов, GUI-интерфейса (Graphical User Interface — графический интерфейс пользователя). Тем не менее, для программ, реализующих реальные задачи, всегда характерна определенная специфика входных данных, какие-то комбинации входных последовательностей или ветвление кода, до проверки которых просто не доходят руки.

**МОЖНО** ЛИ ВЫПОЛНИТЬ ИСЧЕРПЫВАЮЩЕЕ ТЕСТИРОВАНИЕ ПРОГРАММЫ!

первое, что потребуется предпринять, прежде чем приступить к проектированию тестов, — это определить общий объем работ по тестированию. Нужно ли попытаться вы-Выполнить исчерпывающее тестирование программного обеспечения, или же существуют фундаментальные ограничения относительно того, что можно ожидать от тестовых работ? В данной врезке мы проведем анализ примера компонента программного обеспечения, исходя из поставленного выше вопроса.

Рассмотрим программный компонент, который вычисляет стоимость перевозки некоторого продукта с известным весом по заданному почтовому индексу получателя. В программе имеется GUI-интерфейс, через который можно вводить почтовый индекс (наряду с другими данными, имеющими отношение к обрабатываемой транзакции). В этом случае расходы на перевозку представляются промежуточными результатами вычислений, которые пользователю не отображаются. Для того чтобы увидеть результаты вычислений, потребуется отправить запрос в базу данных.

Казовая идея представлена на блок-схеме, изображенной на рис. 3.2. На этой диаграмме показано лишь отношение, связывающее ввод и вывод, поэтому не придется вникать h подробности, относящиеся к GUI-интерфейсу и базе данных.

Для простоты положим, что в качестве стандартного почтового индекса может использоваться любое пятизначное положительное целое число. Другими словами, входные величины принимают значения из диапазона от 00000 до 99999.

Для каждого входного значения программа вычисляет стоимость перевозки, которая колеблется в пределах от $5 до $20. Отображение входа на выход описывается отношением "многие-к-одному", а это означает, что несколько почтовых индексов могут породить одну и ту же стоимость перевозки.

Теперь, когда получено представление о том, как работает эта часть программного обеспечения, можно вернуться к вопросу о возможности исчерпывающего тестирования рассматриваемой программы. В данном случае мы конкретно хотим знать, можно ли проверить эту программу на всех возможных значениях входных данных.

Существуют два класса ввода: допустимый и недопустимый. Пятизначные положительные целые числа суть допустимый ввод, в то же время отрицательные целые числа, числа с десятичной точкой, буквы алфавита, управляющие коды и пробелы не рассматриваются как допустимый ввод.

Простой подсчет количества допустимых почтовых индексов говорит о том, что числом допустимых вводов будет 100000 (пятизначные числа, не разрешенные почтовой службой, игнорируются). Предположим, что автоматизация в данном случае не применяется, т.е. тестирование программы выполняется в неавтоматизированном режиме. Процедура тестирования требует от тестировщика вручную вводить допустимые входные значения, отправлять запросы в базу данных для получения стоимости перевозки, сравнивать вычисленные значения стоимости перевозок и фиксировать полученные результаты. Можно запросто потратить 3 минуты на один ввод, так что на проверку всех допустимых входных значений будет затрачено 5000 часов, что составляет более 2 лет напряженной работы. Кроме того, подобную проверку, возможно, придется выполнить несколько раз, поскольку каждая новая версия должна тестироваться совершенно аналогично.

По сути дела, все возможные входные значения проверить просто невозможно. Картина меняется, если реализовать автоматизацию рассматриваемого вида тестирования, но даже и в этом случае вряд ли захочется тратить время на автоматизацию и тестирование каждого возможного ввода. В главах 4 и 11 будет показано, как сузить диапазон тестовых входных данных и получить компактный, управляемый набор эквивалентных значений с помощью методики разбиения на классы эквивалентности.

Если дополнительно принять во внимание и все недопустимые значения, диапазон еще больше расширится. Можно также запланировать проверку ввода на предмет недопустимости для отрицательных целых чисел, чисел с десятичной точкой, нечисловых значений. Это еще больше увеличит количество возможных вводов, поэтому вероятность выполнения 'абсолютно всех таких попыток станет ничтожно низкой. На основе проведенных рассуждений мы приходим к фундаментальному ограничению тестирования:

Ограничение **тестирования:** Исчерпывающее тестирование компьютерной программы невозможно.

Занимаясь аналогичными исследованиями, Прессман в [43] привел в качестве примера программу на языке С, состоящую из 100 строк с двумя вложенными циклами, причем каждый цикл выполняется от 1 до 20 раз. Во внутреннем цикле находятся четыре конструкций- if-end-else. В своей работе Прессман доказал, что на автоматизированное тестирование этой программы при условии, что на проверку каждого из $10^{14}$ возможных путей будет тратиться одна миллисекунда, потребуется 3170 лет.

Приведенный пример с почтовыми кодами показал, что исчерпывающее тестирование вводов невозможно, а пример Прессмана дает право утверждать, что даже в условиях автоматизированного ввода исчерпывающее логическое тестирование оказывается нереальным. Становится ясно, что одним из непременных условий быстрого тестирования является разумный отбор тестов для выполнения.

*Врезка 3.1*


***Рис. 3.2. Функциональные возможности тестируемой программы***

Поскольку подвергнуть тестированию абсолютно все невозможно, важность выбора того, что нужно протестировать, сомнений не вызывает. Если допустить "перебор" в тестировании, т.е., если тестовое покрытие будет избыточным, то для отладки программного продукта потребуется значительное время, что поставит под угрозу срок сдачи проекта. Если тестирование окажется недостаточным (точнее, недостаточным будет тестовое покрытие), то увеличится риск пропуска того или иного дефекта, устранение которого будет стоить очень дорого, особенно после сдачи программного продукта в эксплуатацию. Отыскать нужный баланс между этими двумя крайностями поможет опыт и способ измерения успешности тестирования.

Вот несколько предложений по разработке стратегии тестирования, которые помогут в поиске оптимального тестового покрытия.

**Тестируйте в первую очередь требования с наивысшим приоритетом.** Предположим, что в вашем распоряжении имеется документ определения требований, в котором требованиям присвоены приоритеты. Выберите те из них, которые представляют для заказчика наибольшую важность, либо которые причинят заказчику наибольшие неприятности в случае выхода программного продукта из строя. Если запланировано тестирование всех требований, и ресурсы это позволяют, естественно, придется тщательно проверить выполнение всех требований. В случае нехватки ресурсов, перед отправкой продукта заказчику необходимо тщательно протестировать требования с наивысшими приоритетами. Возможно, стоит получить согласие заказчика на то, что требования, которые проверены частично или не проверены вообще, не будут поддерживаться вплоть до следующей версии продукта.

**Тестируйте новые функциональные возможности и программный код, который изменялся с целью исправления или совершенствования старых функциональных средств.** Эвристическое правило гласит: если программный код подвергался исправлениям, его необходимо протестировать. В начальной версии программного продукта новым является все. Однако если версия является очередным обновлением или эксплуатационной версией, особое внимание следует уделить новому коду. Имейте в виду, что любые изменения, внесенные в программный код, могут исказить даже те части программы, которые непосредственно "не затрагиваются" изменениями. В этом случае лучше всего как можно чаще выполнять регрессионные тесты для всей функциональности программы, какими бы ни были изменения в коде.

**Используйте разбиение на эквивалентные классы и анализ граничных значений для снижения трудозатрат на тестирование.** Эти технологии описаны в главах 4 и 10; обе технологии могут использоваться для выявления максимального

тестового покрытия за счет определения поднабора входных значений во время тестирования компонентов ввода/вывода.

**Тестируйте те участки, в которых наиболее вероятно присутствие проблем.** Известная пословица гласит: "ошибки имеют свойство накапливаться". Если обнаружен какой-то дефект, очень часто рядом притаились еще несколько дефектов, ждущих своей очереди быть обнаруженными. Если во время анализа требований некоторые участки вызывают особое беспокойство, или есть сведения от разработчиков, что часть компонентов породили массу проблем во время модульного тестирования и проверки взаимодействия и функционирования компонентов, упомянутые участи кода необходимо пометить, дабы обратить на них пристальное внимание при системных испытаниях.

**Сосредоточьте свое внимание на функциях и конфигурациях, с которыми наиболее часто будет иметь дело конечный пользователь.** Если в спецификации требований применяется методика случаев использования или же если у вас имеется доступ к функциональному разрезу конечного пользователя (т.е. математическому ожиданию использования каждой функции), вы получаете дополнительный источник информации для установки приоритетов тестирования. Большая часть времени, отведенного на тестирование, должна затрачиваться на проверку наиболее часто используемых конфигураций и операций. Проблемы тестирования множественных конфигураций будут подробно обсуждаться в разделе "Тестовые конфигурации" далее в этой главе. „

Один из дополнительных подходов придать системному тестированию большую эффективность состоит в том, чтобы убедить разработчиков тщательно выполнять модульное тестирование и проверку взаимодействия и функционирования компонентов, прежде чем передавать программный код тестовой группе. Слишком часто передача тестовой группе той или иной программной конструкции завершается констатацией о невозможности ее установки, или тем, что после установки конструкция не работает. Значительную часть времени на тестирование можно сэкономить, если ввести в действие некоторые критерии приемки, не допускающие передачу тестовой группе неработающих программных кодов.

Объем тестирования должен определяться документами, регламентирующими план проведения испытаний. Пример формата плана проведения испытаний приводится далее в главе. Пример плана проведения испытаний можно найти в третьей части.

## Определение подхода к тестированию

Второй раздел формулировки стратегии тестирования касается определения похода к тестированию. Построение подхода к тестированию начинается с исследования каждой стадии жизненного цикла разработки с целью отбора тестов статического и динамического тестирования, которые могут быть использованы на соответствующей стадии. При этом не имеет значения, какая модель жизненного цикла разработки используется: каскадная, спиралевидная или модель с итеративными версиями — для отбора эффективных тестов можно исследовать этапы любой перечисленной модели. В качестве примера возьмем каскадную модель и выясним, какие виды тестирования могут для нее использоваться.

**Стадия формулирования требований.** Определите все документы, которые содержат требования и генерируются на данной стадии, а также все документы, содержащие итоги инспекций, которые получены как результат статического тестирования. Если вы составляете план проведения испытаний на стадии формулирования требований, этот шаг может оказать помощь при составлении плана и графика выполнения необходимых инспекций. Назначьте членов группы тестирования, которые должны принимать участие в проведении инспекций. Выберите хранилище для результатов инспекций.

**Стадия системного проектирования.** Определите все проектные документы, которые составляются на данной стадии и план участия группы тестирования в связанных с этим инспекциях. В документах, формулирующих требования (документы технического задания), указано, *что* должно тестироваться, а документы эскизного проекта помогут получить представление о том, *как* проводить тестирование.

**Стадии тестирования проектов программ, программных кодов, модульного тестирования и комплексных испытаний.** Если группа тестирования принимает участие в статическом или динамическом тестировании любых из указанных выше видов деятельности, потребуются соответствующие планы. Время от времени, группа тестирования прогоняет тесты с целью выявления утечек памяти или проверки цикломатической сложности тестирования на стадиях модульного тестирования и комплексных испытаний. Если этот так, то планы таких испытаний должны учитываться в общем плане испытаний. Если ответственность за эти стадии возлагается только на коллектив разработчиков, их роль должна определяться в форме предположения.

**Системные испытания.** Опишите запланированные вами виды тестирования: функциональная проверка, нагрузочные испытания, испытания под нагрузкой, проверка безопасности, проверка наращиваемости, проверка удобства и простоты обслуживания и другие. Сформулируйте высокоуровневые цели для этих видов тестирования и укажите, как проследить их связь с породившими их требованиями и функциональными спецификациями. Если какие-то виды тестирования опускаются, дайте логическое обоснование их пропуска и оцените связанные с этим риски.

**Приемочные испытания.** Дайте описание плана приемочных испытаний, включая альфа-, бета- и другие виды тестирования. Часто бывает полезно составить отдельный план приемочных испытаний, а в нем указать объемы работ и ограничения, накладываемые на тестирование, критерии включений и исключения из испытаний и условия испытаний. Поскольку нередко цель приемочных испытаний состоит не только в том, чтобы продемонстрировать заказчику функциональные возможности программного продукта, но и в обнаружении дефектов, план приемочных испытаний целесообразно рассматривать как документ, направленный извне.

**Регрессионное тестирование.** Укажите в плане регрессионного тестирования, составлен ли он для эксплуатационных версий или же является частью стратегии итеративного выпуска версий. Опишите, как выбираются и разрабатываются регрессионные тесты, какое должно использоваться оборудование й какова страте-

гия автоматизации регрессионного тестирования. Исходя из предположения, что тестами покрываются не все функциональные возможности, укажите объемы регрессионного тестирования и риски, связанные с тем, что некоторые участки будут пропущены и останутся непокрытыми.

Подход к тестированию должен отражаться в документах, содержащих планы проведения испытаний. Один из способов показан на примере шаблона далее в этой главе. Пример плана проведения испытаний для приложения TMT приводится в третьей части книги.

## Определение критериев тестирования и точек контроля качества

Цель определения критериев тестирования заключается в установке на месте заказчика набора правил, регламентирующих поток тестирования. Очень важно определиться с критериями тестирования до начала испытаний. После того, как вы приступили к выполнению плана испытаний, поздно объявлять критерии, инициирующие или останавливающие тестирование. Например, если вы рассчитываете на то, что программный продукт успешно пройдет модульное тестирование и комплексные испытания, вы должны заявить об этом заранее в подготовленном плане испытаний. После этого план должен быть проверен и утвержден людьми, которые будут заниматься модульным тестированием и комплексными испытаниями. Ввод в действие и выполнение критериев тестирования помогает проекту выдерживать намеченные сроки за счет исключения временных потерь, которые будут неизбежны в случае, когда на испытательный стенд попадает программный продукт, не готовый к испытаниям, либо в ситуации, когда продолжаются испытания продукта, который нужно отправлять на переделку.

Поскольку критерии тестирования затрагивают интересы других групп, они не могут односторонне устанавливаться группой тестирования. Применение критериев должно быть согласовано со всеми заинтересованными сторонами. Критерии могут быть установлены для модульного тестирования и комплексных испытаний, однако такие критерии обычно определяются организацией-разработчиком. Рассматриваемые здесь критерии тестирования применяются главным образом в отношении системного тестирования.

Существует пять типов критериев, которые могут определяться перед началом системного тестирования.

**Критерий входа** описывает, что нужно сделать перед началом тестирования. Например, возможно, потребуется располагать документами технического задания в окончательной форме. Можно поставить задачу, чтобы программный продукт был упакован в том же виде, в каком он будет поставляться заказчику. Во время тестирования может возникнуть необходимость в служебных программах, конфигурационных файлы или данных, которыми будет пользоваться заказчик. Если на вас возложена ответственность за проверку документации, сопровождающей программный продукт, она также должна быть доступна во время тестирования. Один из способов выполнения критерия входа предусматривает проверку готовности к испытаниям. Эта проверка использует контрольную таблицу элементов действий, которые другие группы согласились передать как входные данных для тестирования.

**Критерий выхода** описывает то, что вы считаете необходимым для завершения испытаний. Несмотря на то что группы тестирования часто пытаются сделать критерий выхода условием поставки программного продукта, это нереально. Решение на поставку принимается и должно приниматься высшим руководством разработок. Критерий выхода из испытаний должен звучать примерно так: "все запланированные тесты выполнены, все исправленные дефекты проверены, уведомления о всех обнаруженных новых дефектах были выданы. Невыполненные пункты плана, такие как неудачный прогон некоторого набора тестов из-за неисправности оборудования, задокументированы". Как и в случае критерия входа в испытания, допускается проведение проверки готовности, которая обеспечит полное выполнение всех тестов, и оценки готовности программного продукта к поставкам на базе результатов тестирования.

**Критерий приостановки/возобновления** описывает, что произойдет, если по причине из-за дефектов продолжение тестирования окажется невозможным. Другими словами, если дела складываются настолько неудачно, что запланированные испытания провести не удается, их необходимо прекратить до тех пор, пока обнаруженные дефекты не будут устранены.

**Критерий успешного/неудачного прохождения теста.** Прогон каждого теста должен давать заранее известные результаты. Если получен ожидаемый результат, считается, что продукт успешно прошел тест, в противном случае прохождение теста завершается неудачно. В то же время может случиться так, что прогон некоторой группы тестов не выполняется, поскольку они либо искажены или заблокированы дефектами, либо необходимые для их прогона ресурсы отсутствуют. Целесообразно определить заранее, что делать с тестами, которые не удалось выполнить. Возможно, будет запланировано пометить каждый невыполненный тест буквой "N" в итоговом отчете и объяснить в поле комментария, что случилось и что было предпринято в контексте решения проблемы. Может быть, в ваши планы входит замена искаженных тестов специальным видом тестирования и регистрация результатов в специальном акте об испытаниях.

**Другие критерии, определяемые процессом или стандартами.** Если программный продукт должен соответствовать некоторому стандарту или ваша компания предъявляет определенные требования к выполняемому процессу, скорее всего, придется учесть ряд дополнительных критериев. Например, в вашем локальном процессе могут быть определены конкретные средства, выдающие сообщения об обнаруженных дефектах или отчеты по результатам испытаний.

В дополнение к приведенным выше критериям, полезно определить критерий тестирования и подготовить контрольную таблицу готовности в рамках плана проведения испытаний. Этот момент отображен в описании плана проведения испытаний далее в главе и в примере плана проведения испытаний в третьей части книги.

## Определение стратегии автоматизации

При наличии реальных планов и разумных предположений использование автоматизированных инструментальных средств и автоматизированных тестовых случаев представляет собой прекрасный способ снижения временных затрат на тестирование программного продукта. Любая многократно выполняемая задача является кандида-

том на автоматизацию. Однако обычно на автоматизацию задачи уходит намного больше времени, чем на ее выполнение, поэтому для каждой задачи, которая может быть автоматизирована, целесообразно провести тщательный анализ потенциального выигрыша от автоматизации. Выполняя анализ возможных выгод, следует помнить, что для самой автоматизации характерен собственный автономный жизненный цикл. Эффективная автоматизация требует специальной подготовки персонала, разработки, отладки и верификации, как и любой другой проект разработки программного обеспечения. Бесплановая и плохо выполненная автоматизация означает не только напрасный расход ресурсов, она даже может привести к нарушению графика выполняемых работ, если время будет тратиться на отладку средств автоматизации, а не на тестирование.

Дастин (Dustin), Рашка (Rashka) и Пауль (Paul) в [15] приводят наиболее распространенные необоснованные предположения, связанные с применением автоматизированного тестирования, равно как и реальные выгоды, на которые можно рассчитывать, если автоматизации выполнена правильно и следует строгому процессу. Необоснованные предположения (ожидания) сведены в табл. 3.1, а достижимые выгоды от автоматизации перечислены в табл. 3.2.

Если вы все-таки решились вложить средства в автоматизацию тестирования, следует проанализировать все последствия этого решения в рамках выбранной стратегии тестирования. Например, задача формирования тестовой среды может зависеть от того, как будет осуществляться прогон тестов — в автоматическом или в ручном режиме. Поскольку и для задач, решаемых в ручном режиме, и для задач, решаемых в автоматическом режиме, необходимо приобретать одни и те же аппаратные средства, соединять их между собой кабелями, подключать к компьютерным сетям и вводить их в эксплуатацию, возможно, имеет смысл сделать автоматизированную тестовую среду постоянно действующей конфигурацией. Подобный подход позволит выполнять автоматизированные тесты без вмешательства со стороны оператора. Если предполагается выполнять прогон автоматизированных тестов в рамках регрессивных испытаний или для целей технического обслуживания, имеет смысл построить специализированную стационарную испытательную установку, которая будет находиться на рабочей площадке весь период, пока поддерживается программный продукт. Однако такое решение влечет за собой существенное увеличение расходов на установку соответствующих аппаратных средств и на помещения, в котором располагается испытательный стенд. В случае неавтоматизированного режима тестирования конфигурация технических средств создается по ходу дела.

Другая проблема, связанная с разработкой стратегии автоматизации, имеет отношение к процессу. Хорошо известно, что тестирование программного обеспечения должно быть сформированным до принятия решения по поводу автоматизации. Термин "сформированный" означает, что тестирование интегрировано в жизненный цикл программного обеспечения, что в основу целей испытаний и тестовых случаев положены требования, и что существует автономная организация тестов. Попытка использовать автоматизацию в условиях хаотичного процесса тестирования едва ли окажется успешной. Автоматизацию лучше всего внедрять как часть непрерывного процесса совершенствования процесса тестирования, но не применять подход "большого взрыва", требующий автоматизации всего и сразу. (Более подробную информацию, касающуюся совершенствования процесса тестирования, можно найти в главе 6).

**Таблица 3.1. Необоснованные ожидания от автоматизированного тестирования (по материалам [15])**

| Необоснованные ожидания | Комментарий |
|---|---|
| Автоматизировать можно все, что угодно | Можно привести несколько примеров, когда автоматизация не имеет смысла. Если тот или иной тест нужно прогнать только один раз, нет смысла автоматизировать его, поскольку на автоматизацию уйдет больше времени, чем на прогон теста вручную. Если требования сформулированы нечетко или программные коды регулярно подвергаются изменениям, то не имеет смысла автоматизировать тесты, которые будут постоянно меняться. Кроме того, нецелесообразна также автоматизация тестов, требующих вмешательства оператора, подобных, например, замене оборудования во время прогона теста. |
| Все необходимое может выполнить единственное инструментальное средство | В настоящее время ни одно из доступных на рынке программных средств не способно генерировать планы проведения испытаний, поддерживать проектирование тестов и управлять исполнением тестов. Не существует ни одного инструментального средства, которое поддерживало бы все операционные системы и все языки программирования. Средства автоматизации следует тщательно выбирать под имеющуюся задачу. |
| Временной график тестирования немедленно сократится | Автоматизация тестирования может уменьшить время, затрачиваемое на тестирование, однако для того, чтобы сократить график работ, необходимы определенные затраты на подготовку персонала, разработку автоматизации и совершенствование процесса тестирования. После принятия решения об автоматизации график выполнения работ расширяется ввиду необходимости поддержки начальных вложений ресурсов. |
| Современные средства автоматизации просты в использовании | Поставщики инструментальных средств постоянно повышают простоту использования инструментальных средств автоматизации, однако у вас не должно складываться впечатление, что эти средства можно эффективно задействовать без специальной подготовки. При планировании стратегии автоматизации обязательно следует дать адекватную оценку времени, затрачиваемого на такую подготовку, а также поддержки со стороны поставщиков соответствующих программных продуктов. |
| Автоматизация обеспечивает стопроцентное тестовое покрытие | Даже с использованием автоматизации полное тестирование компьютерной программы невозможно. Подробное обоснование этого тезиса приводится во врезке 3.1. |

**Таблица 3.2. Выгоды, которые приносит автоматизированное тестирование (по материалам [15])**

| Извлекаемая выгода | Комментарий |
|---|---|
| **Повышенная надежность системы** | |
| Уточненные определения требований | Доступны различные инструментальные средства, которые поддерживают разработку проверяемых требований. Некоторые из этих средств требуют использования формальных языков, в то время как другие моделируют требования графически. |
| Усовершенствованные тесты производительности и испытания при перегрузках | Существуют инструментальные средства тестирования под нагрузкой, позволяющие специалистам по тестированию разрабатывать сценарии, которые осуществляют автоматический сбор статистических данных по времени отклика, по количеству пользователей, по числу транзакций и продолжительности больших транзакций. Есть возможность проводить испытания при перегрузках, в рамках которых система подвергается предельным нагрузкам с целью определения причины отказа. |
| Повышенная повторяемость тестирования | Автоматизированное тестирование протекает каждый раз одинаково, что существенно уменьшает количество ошибок, вызванных непостоянствами, имеющими место при ручной методике испытаний. Повышенная повторяемость тестирования очень важна при воспроизведении дефектов и проверке исправлений. |
| **Повышение качества тестирования** | |
| Проверочные испытания сборок | Автоматизированные испытания могут использоваться как своего рода "испытания герметичности" программной сборки (build) при ее переходе со стадии разработки на стадию тестирования. Критерии входа часто требуют проведения испытаний на герметичность, благодаря которым к тестированию не допускаются сборки, не готовые к испытаниям. В результате достигается существенная экономия трудозатрат. |
| Повышение качества статического и динамического тестирования кода | Возможен анализ кода на предмет утечек памяти, на чрезмерную сложность, на ошибки во время прогона, на присутствие недостижимых кодов и другие виды дефектов. Более подробную информацию о статическом тестировании можно найти в главе 9. |
| Повышение качества регрессионного тестирования | Автоматизированное регрессионное тестирование используется для проверки, не вкралась ли ошибка при переносе ранее правильно работавшего программного кода в новую сборку. |
| Повышение качества проверки на совместимость | В некоторых случаях автоматизированные тесты можно переносить в другие операционные системы и платформы, что расширяет тестовое покрытие без увеличения объемов неавтоматизированного тестирования. |
| Более эффективное выполнение однотипных задач тестирования и тестирование в нерабочее время | Автоматизация однообразных, однотипных задач уменьшает вероятность ошибок тестирования и позволяет сократить сроки испытаний. Автоматический прогон тестов можно выполнять ночью или в выходные дни, тем самым давая возможность специалистам по тестированию в рабочие часы сосредоточиться на решении более сложных и творческих задач. |

*Окончание табл. 3.2*

| Извлекаемая выгода | Комментарий |
|---|---|
| **Сокращение сроков тестирования** | |
| Уменьшение времени прогона тестов | Единожды разработанные и отлаженные автоматизированные тестовые случаи обычно требуют меньше времени для прогона, чем неавтоматизированные тесты. |
| На анализ результатов тестирования требуется меньше времени | Автоматизированные испытания обычно генерируют журналы и отчеты по испытаниям, |
| Меньшие затраты времени на определение состояния теста и на составление отчетов по испытаниям | Большая часть автоматизированных средств управления тестированием способны быстро генерировать данные о состоянии тестирования, в которых указываются количество выполненных тестов, количество удачно и неудачно завершенных тестов и другие полезные статистические данные. |

Планы относительно автоматизации процесса тестирования должны быть включены в раздел Подход плана проведения испытаний. Если ожидается большой объем трудозатрат на проведение автоматизации, возможно, стоит иметь план автоматизации тестирования в виде отдельного документа, регламентирующего все аспекты проекта автоматизации: анализ за и против с целью выбора компромиссного решения, объем трудозатрат, распределение обязанностей, анализ инструментальных средств и график работ.

# Определение стратегии тестирования

После определения стратегии тестирования можно выходить на следующий уровень детализации процесса планирования испытаний: к определению испытательной системы. Понятие испытательной системы относится не только к аппаратным средствам, необходимым для проведения испытаний, но и к архитектуре тестов и к тестовым конфигурациям. Обсуждение начнем с архитектуры тестов.

## Архитектура тестов

Архитектура тестов имеет отношение к организационной структуре тестовых случаев. На практике целесообразна такая организация тестов, когда каждый тест рассчитан на проверку некоторой четко заданной части функциональных возможностей программного продукта. Если тест закончится неудачей, разработчик легко может повторно прогнать этот тест и воспроизвести обнаруженный дефект, затем проанализировать его и попытаться выявить причину. В идеальном случае границы обследования теста должны быть сужены, а сам тест должен выполнять некоторый минимальный набор действий, необходимых для того, чтобы вызвать отказ системы.

Если основой проводимых испытаний служат требования к программному продукту, то практическим правилом для определения границ обследования конкретного теста может служить критерий, по условиям которого на каждое такое требование полагается, по меньшей мере, один тест. Помимо того, что это правило определяет эффективные размеры тестов, принцип "по меньшей мере, один тест на требование" упрощает поддержку планирования проведения испытаний. На основе анализа тех-

нического задания можно получить представление о том, сколько нужно новых тестов для испытания нового программного продукта. Если составлять тесты, руководствуясь этим принципом, несложно оценить, какое время отымет разработка и прогон тестов, основанных на ранее полученных результатах.

Испытания по описанному выше принципу позволяют систематизировать организацию тестов. Требования обычно объединяются в группы по функциональным признакам. Вы можете организовать свои тесты по такому же принципу. Тесты, осуществляющие проверку готовности к работе технических средств заказчика, могут быть объединены в одну группу; точно так же тесты, проверяющие механизмы установки и наращивания возможностей системы, образуют другую группу. Группа родственных тестов называется *тестовым набором (test suite)*.

Каждое требование может состоять из нескольких компонентов. Например, требование, регламентирующее процедуру редактирование элементов базы данных конг кретного заказчика, может определять несколько различных сценариев. *Тестовый случай (test case)* есть набор входных данных тестов, условий выполнения тестов **и** ожидаемых результатов, который ориентирован на достижение конкретной цели. Тестовый случай представляет собой минимальный тестовый модуль, допускающий независимое выполнение от начала до конца. Отношения, связывающие тестовые наборы, тесты и тестовые случаи, показаны на рис. 3.3 и в табл. 3.3.

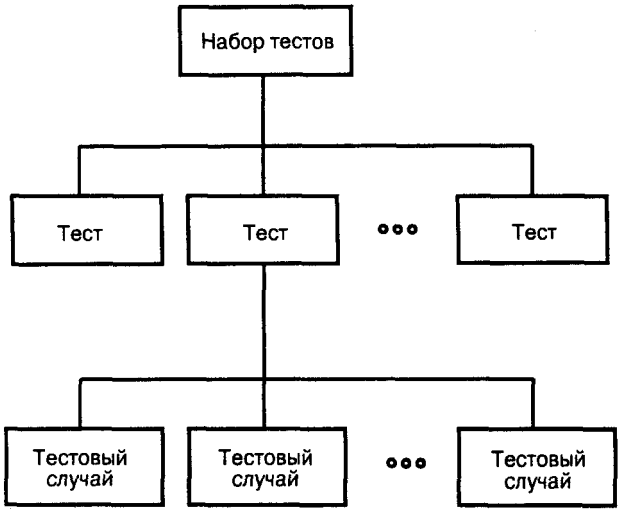



***Рис. 3.3. Архитектура тестов***

После того, как удалось определить организационную структуру тестов, возникает необходимость дать описание хранилища для хранения тестов **и** средств управления версиями для последующего воспроизводства тестов. Это хранилище должно обладать определенными характеристиками:

- Специалисты по тестированию, разрабатывающие и осуществляет прогон тестов, должны иметь простой доступ к хранилищу. Часто в качестве хранилища используется сетевой дисковый накопитель коллективного доступа.

- С этого хранилища регулярно снимаются копии, а полученные резервные копии файлов периодически восстанавливаются с целью проверки правильности функционирования средств копирования и восстановления. Необходимость дублирования всех планов проведения испытаний, тестовых случаев, результатов тестирования очевидна, тем не менее, зачастую такое дублирование игнорируется.
- Должно быть налажено управление версиями, что обеспечит простое выявление более старых тестов. Управление версиями при необходимости обеспечивает воспроизведение любого теста, который когда-либо выполнялся на программном продукте.

**Таблица 3.3. Иерархия тестов**

| Уровень | Определение |
|---|---|
| **Набор** тестов | Совокупность тестов, используемых для проверки связанных групп требований и функций. |
| **Тест** | Один или большее число тестов, которые предназначаются для проверки выполнения одного требования или функции. |
| Тестовый случай | Наименьший тестовый модуль, который может выполняться самостоятельно от начала до конца. |

## Инструментальные средства тестирования

В начале этой главы отмечалось, что решение перейти на автоматическое тестирование оказывает влияние на всю стратегию тестирования. В этом разделе основное внимание будет уделяться инструментальным средствам тестирования. Общая стратегия автоматизации тестирования не ограничивается лишь одним программным продуктом; эта стратегия должна быть глобальной, распространяющаяся на многие проекты, и тщательно интегрированной с процессом тестирования.

Таким образом, план проведения испытаний конкретного программного продукта определяет, какой фрагмент общей стратегии автоматизации будет реализован в текущем проекте. Возможно, потребуется добавить автоматизированное испытание программного продукта под нагрузкой, либо автоматизировать тестирование GUI-интерфейса. Может быть, возникнет желание воспользоваться автоматизированным генератором отчетов, который пересылает результаты тестирования и метрики программного продукта на Web-страницу. После определения затрат времени и средств, связанных с оценкой и приобретением (или построением) инструментария, который необходим для завершения автоматизации, и последующей оценки трудозатрат на обучение новому инструментальному средству, скорее всего, будет принято решение о включении в проект только одного-двух инструментальных средств подобного рода. Включение большего числа новых инструментальных средств мало того, что непродуктивно, но еще и требует существенных затрат.

Существует широкое разнообразие инструментальных средств, служащих для поддержки тестирования. Некоторые из этих средств перечислены в табл. 3.4. Эта таблица отображает инструментальные средства на этапы жизненного цикла разработки, где они могут использоваться.

Более подробную информацию по инструментальным средствам тестирования можно найти в [15], [30] и [40]. Каждый перечисленный источник содержит также полезную информацию, связанную с планированием испытаний.

**Таблица 3.4. Характерные инструментальные средства тестирования**

| Инструментальное средство тестирования | Помогает тестировщику выполнить: | Стадия жизненного цикла |
|---|---|---|
| Средство отслеживания дефектов | Вводить и сохранять сообщения о дефектах, генерировать метрики программного продукта и итоговые отчеты. | Все стадии |
| Средства организации испытаний | Сохранять планы проведения испытаний, тестовые случаи (ручные или автоматизированные), результаты тестирования.<br>Уведомлять о состоянии тестирования. | Все стадии |
| Средства организации требований | Организовывать и отслеживать требования. Прослеживать тесты до породивших их требований | Соответствующие данные первоначально вводятся на стадии анализа и оценки, но могут быть использованы на протяжении всего жизненного цикла |
| Средства обнаружения утечек памяти и ошибок во время выполнения программы | Прогон тестов для обнаружения утечек памяти и ошибок во время выполнения программы. | Обычно используется на этапе тестирования программных кодов, модульного тестирования и проверки взаимодействия и функционирования компонентов системы |
| Инструментальное средство тестирования исходного кода | Проверка сложности, цикломатической сложности и соответствия стандартам. | Обычно используется на этапе тестирования программных кодов, модульного тестирования и проверки взаимодействия и функционирования компонентов системы |
| Анализаторы программных кодов и тестового покрытия | Использование инструментальных средств контроля для измерения фрагмента программного кода, покрытого средствами динамического тестирования. | Используется в рамках динамического тестирования на стадии на стадиях проверки взаимодействия и функционирования компонентов или системных испытаний |
| Блок сравнения исходных кодов | Сравнение двух версий исходного кода на предмет их идентичности; если коды не идентичны, определяются их различия. | Системные испытания и регрессивное тестирование |

*Окончание табл. 3.4*

| Инструментальное средство тестирования | Помогает тестировщику выполнить: | Стадия жизненного цикла |
|---|---|---|
| Генераторы тестовых данных | Генерация тестовых данных и заполнение баз данных. | Все этапы тестирования: модульное тестирование проверка взаимодействия и функционирования компонентов, системные и регрессивные испытания |
| Инструментальные средства реализации сценариев для GUI-интерфейсов | Запись выполненных пользователем нажатий клавиш и щелчков кнопками мыши, воспроизведение этих пользовательских действий, сравнение результатов тестирования с ожидаемыми результатами. | Системные и регрессивные испытания |
| Инструментальные средства, обеспечивающие загрузку, измерение рабочих характеристик и предельные нагрузки системы | Одновременное выполнение нескольких клиентских программ с целью нагрузки сервера.<br>Выполнение клиентских программ в условиях предельных нагрузок с целью определения момента выхода из строя системы. | Системные и регрессивные испытания |
| Сетевые инструментальные средства тестирования | Оценка способности сервера и сети безошибочно передавать данные. | Системные и регрессивные испытания |
| Средства написания сценариев командной строки | Автоматизация команд для настройки испытательной установки, загрузки тестовых данных или анализа результатов тестирования . | Все этапы тестирования: модульное тестирование проверка взаимодействия и функционирования компонентов, системные и регрессивные испытания |

## Среда тестирования

Среда тестирования состоит из физических средств тестирования, операционной системы, под управлением которой выполняется программный продукт, и вычислительной платформы, на которой эксплуатируется программный продукт. Многие компании, занимающиеся тестированием программных продуктов, обзавелись собственными испытательными лабораториями. Как и автоматизация, развертывание хорошей испытательной лаборатории связано с долгосрочными инвестициями, которые ложатся на множество проектов и требуют отдельных усилий. Разумеется, хорошего качества тестирования можно добиться и на одном компьютере, установленном в отдельном крохотном помещении, однако общепринятой нормой стало создание испытательных лабораторий, обеспечивающих настройку тестовой среды и условий, в рамках которых производится оценка качества программного продукта.

Рекс Блек (Rex Black) в [33] развернул дискуссию вокруг вопросов оснащения и организации испытательных лабораторий. Он предлагает следующее: прежде чем принимать решение относительно необходимости создавать специализированную испытательную лабораторию, потребуется дать ответы на приводимые ниже вопро-

сы. К созданию специализированной испытательной лаборатории можно приступать, если вы ответите "да" на хотя бы один из следующих вопросов:

- Существует ли необходимость в крупногабаритном стационарном оборудовании, таком как, например, камера искусственного климата, генераторы данных и анализаторы или приборы измерения электромагнитного излучения?
- Должны ли предприниматься специальные меры безопасности по защите конфиденциальной информации или крупных вложений?
- Нужно ли тщательно следить за условиями, в которых проводятся испытания, и регулировать их? Это может, например, означать, что персонал, не принимающий непосредственного участия в испытаниях, не должен иметь доступа к системным испытаниям во имя предотвращения изменений в условиях испытаний.
- Требуется ли поддерживать некоторый набор фиксированных тестовых конфигураций в течение продолжительного периода времени, например, при поддержке итеративного набора версий?

Планируя тестирование конкретной версии программного продукта, следует определить, какой аспект общей стратегии построения испытательной лаборатории нужно реализовать. Например, может возникнуть необходимость добавить определенное оборудование, генерирующее данные, либо дополнительные платформы для испытания программного продукта под нагрузкой или при перегрузках.

## Конфигурации аппаратных средств тестирования

Предположим, что тестируемый программный продукт, в соответствии со спецификацией, должен выполняться на широком наборе сетевых конфигураций, использует несколько различных конфигураций клиентских машин, должен работать под управлением различных операционных систем и быть доступным из разных браузеров. После исследований числа возможных конфигураций оборудования, могут быть получены, например, такие комбинации:

- 5 сетевых конфигураций
- 10 сочетаний браузеров и операционных систем
- 20 комбинаций клиентских конфигураций (центральный процессор, жесткие диски, видеоадаптеры и периферийные устройства)

Таким образом, количество возможных тестовых конфигураций есть 5 хЮх20=1000. Если вы рассчитываете на то, что два специалиста по тестированию способны провести запланированные системные испытания на одной конфигурации, то для полного завершения тестирования всех конфигураций вам потребуются 240000 человеко-часов, или 125 человеко-лет! Если необходимые средства и большой штат специалистов по тестированию отсутствует, либо же если просто нет достаточного времени, подобное тестирование не имеет смысла.

В начале главы мы сформулировали основной принцип тестирования, который гласит: исчерпывающее тестирование программы провести невозможно. То же самое справедливо и в случаях, когда приходится выполнять тестирование нескольких комбинаций возможных конфигураций системы: невозможно полностью протестировать

абсолютно все конфигурации системы. Основной способ состоит в установке соответствующих приоритетов конфигурациям. А вот дальше уже можно решать, какие конфигурации нужно подвергать полной отладке, а к каким применять частичное тестирование.

Установка приоритетов для конфигураций обычно зависит от следующих факторов:

- Частота использования: сколько экземпляров заданной конфигурации скорее всего будет использоваться?
- Риск отказа в работе системы: существуют ли ответственные конфигурации для важных заказчиков?
- Вероятность отказа системы: фиксировались ли в прошлом отказы конкретных конфигураций?

После определения конфигурации для полного или частичного тестирования следующим действием должно быть определение, какие тесты необходимо выполнить на конфигурациях, подлежащих частичному тестированию. Предполагая, что приоритеты тестам уже присвоены (см. раздел "Определение объемов тестовых работ" ранее в главе), можно выполнять прогон только тестов с высокими приоритетами на некоторых конфигурациях и тестов с высокой вероятностью отказа на конфигурациях, на которых имели место проблемы в прошлом.

После идентификации тестируемых конфигураций потребуется определить их во всех деталях, чтобы специалисты по тестированию могли их установить и воспроизвести в момент, когда возникнет необходимость прогона тестов. Каждая конфигурация средств тестирования может быть определена при помощи блок-схемы и спецификации запасных частей или, если связность между компонентами системы очевидна, конфигурацию можно задать списком компонент.

## Оценка трудозатрат на тестирование

Одной из наиболее важных компонент планирования является оценка трудозатрат и времени, необходимых для тестирования. Затраты на тестирование могут составлять существенную часть сметной стоимости проекта, при этом жизненно важно для успеха этой операции, чтобы тестирование проводило достаточное число специалистов и у них было достаточно времени на качественное выполнение своих задач.

Получение оценки трудозатрат на выполнение проекта можно разбить на пять этапов:

1. Определение задач, которые должны быть выполнены. Эта оценка начинается с определения работ, которые необходимо выполнить для того, чтобы тестирование программного продукта считалось состоявшимся. В рамках некоторых методологий на этом этапе вполне достаточно разбить работу на задачи. Если используются менее формальные методы, то результатом этого этапа может быть простой список задач.
2. Оценка трудозатрат на решение отдельных задач и всего жизненного цикла тестирования. Каждая задача, выявленная на первом этапе, требует для своего решения определенных трудозатрат, представляющих собой объем работ, необходимых для выполнения соответствующей задачи. Трудозатраты

представлены в виде произведения количества исполнителей на затраченное ими время и измеряется в таких единицах как человеко-день или человеко-месяц. Существуют различные методы оценки трудозатрат, часть из которых рассматривается далее в разделе.

3. **Определение времени, требуемого для решения каждой задачи и длительности всего жизненного цикла тестирования.** Время, необходимое для решения задачи, измеряется в днях, неделях или месяцах. Время, необходимое для выполнения той или иной задачи, зависит от количества исполнителей, но как будет показано ниже, эта зависимость не обязательно линейная. Суммарная продолжительность работ по тестированию зависит от продолжительности решения отдельных задач, однако это не простое суммирование, поскольку некоторые задачи можно решать одновременно с другими.

4. **Построение подробного расписания и поэтапного графика каждой тестовой задачи.** На основе результатов трех предыдущих этапов можно построить график выполнения работ, возможно, в виде диаграммы Ганта (Gantt), и вычислить сумму наиболее важных временных значений.

5. **Оценка рисков невыполнения графика работ и формулировка планов их снижения.** Примите во внимание проблемы, которые могут возникнуть при решении задач за запланированные промежутки времени, и предусмотрите средства решения этих проблем.

Прежде чем приступить к более подробному анализу перечисленных шагов, важно понять, насколько точными могут быть наши оценки. В самом начале проекта (перед выявлением и изучением требований) очень трудно определить, какой окажется стоимость проекта. Фактическая стоимость проекта становится известной только на завершающей стадии проекта. На начальной стадии проектирования может иметь место как недооценка, так и переоценка фактической стоимости проекта, причем приблизительно в четыре раза. После того, как все требования будут проанализированы и начальная фаза планирования завершена, оценка стоимости проекта обычно отличается от окончательной, фактической стоимости проекта примерно в два раза [40]. Точность оценки иллюстрируется графиком, показанным на рис. 3.4.

Ввиду неточностей, свойственных процессу оценки, целесообразно периодически возвращаться к оценке трудозатрат и графика выполнения работ и считать это неотъемлемой частью организации работ по тестированию. Если затраты времени и трудозатраты начинают превышать исходные оценки, желательно как можно быстрее поставить в известность об этом факте руководство проекта, а не ждать, когда эта ситуация переродится в настоящий кризис.

Следующие несколько разделов данной главы дают обзор процесса оценки. Предлагаются альтернативные стратегии выполнения трех основных действий из приведенного выше списка, причем вместе со всеми за и против. Более подробно технологии оценки, особенно те из них, которые относятся к категории алгоритмической оценки, будут рассматриваться в главе 12. Хорошим справочным пособием по оценке является уже ставшая классикой [40]. Кроме того, внимание заслуживает также и относительно недавняя публикация [33], в которой описывается модель оценки COCOMO II.

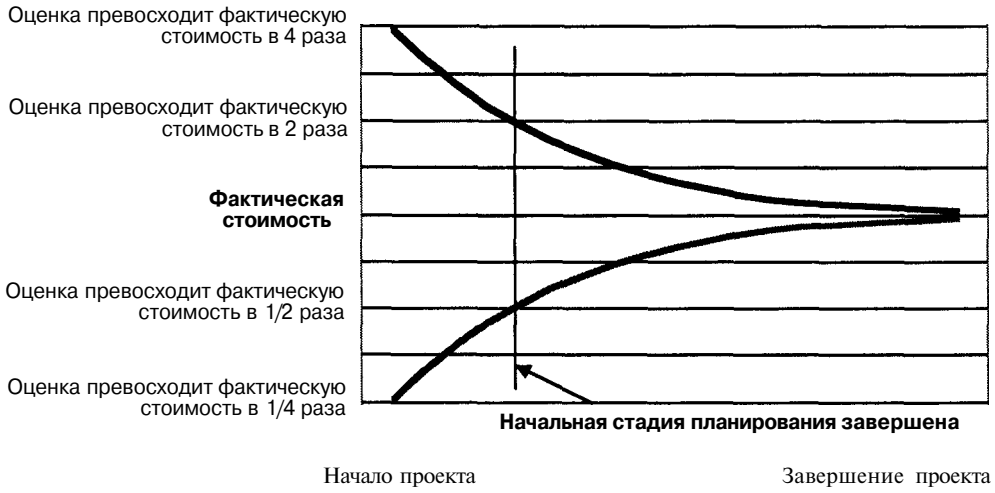



***Рис. 3.4. Точность оценки. По материалам [40]***

## Определение задач

Простейший способ получения списка задач предусматривает просмотр документов, в которых сформулированы требования, и выписывание всех выполняемых работ, чтобы затем проверить функции, определенные этими требованиями. Как только список будет определен, составляющим его задачам присваиваются приоритеты. В качестве руководящих принципов в отношении приоритетов функций должны использоваться документы, содержащие формулировки требований. В то же время могут возникать и дополнительные соображения, оказывающие влияние на выбор приоритета. Например, некоторые из программных кодов, реализующих ту или иную функцию, при переходе на новую версию программного продукта могут претерпевать лишь незначительные изменения, тогда как другие функции в новой версии появляются впервые. По-видимому, новому программному коду имеет смысл посвятить повышенное внимание, нежели модифицированному коду. Кроме того, придется уделить больше времени на проверку проектных решений и структуры, а также на отладку нового кода.

В дополнение к обычному просмотру требований необходимо учесть задачи, которые не имеют прямого отношения к документам с требованиями, но то же время делают тестирование возможным. Например, неплохо было бы включить в указанный выше список такие задачи:

- Составление плана проведения испытаний
- Пересмотр плана проведения испытаний
- Разработка тестовых случаев
- Отладка тестовых случаев
- Проверка исправления дефектов
- Определение тестов и показателей программного продукта и процесса их сбора и использования

- Статическое тестирование: проверка и пересмотр проектов программных средств и кодов
- Реализация выбранной стратегии автоматизации тестирования в разумных пределах
- Пересмотр пользовательской документации
- Приобретение оборудования, необходимого для реализации данного проекта, как составная часть стратегии создания универсальной испытательной лаборатории
- Прием на работу специалистов по различным профилям тестирования необходимой квалификации
- Освоение специалистами по тестированию новых технологий и новых инструментальных средств
- Прогон тестов и отчеты по результатам тестирования
- Проведение проверок готовности
- Оценка выполнения проекта.

После построения исходного списка полезно его просмотреть вместе с группой тестирования и мысленно пройти через полный цикл разработки жизненного цикла, определяя необходимые компоненты и задачи, которые должны выполняться для нужд группы тестирования. По всем признакам список задач является динамическим документом, претерпевающим изменения на стадии планирования и, скорее всего, на протяжении всего жизненного цикла тестирования по мере роста понимания того, что должно быть сделано.

Список задач имеет еще одно название — список WBS (Work Breakdown Structure — декомпозиция работ). Декомпозиция работ часто менее формальна, чем описанный выше список задач, с другой стороны, она может выглядеть как иерархически упорядоченный список видов деятельности, в которых каждой задаче присваивается "дескриптор" или идентификатор. Дескриптор позволяет отслеживать задачу на протяжении всего жизненного цикла разработки и предоставляет возможность связывать измерения трудозатрат или расходов с различными задачами. Благодаря этому можно снимать наборы показателей, характеризующих фактические затраты на протяжении всего проекта. Разумеется, больше формальностей и больше отчетности означают большие накладные расходы в смысле времени и стоимости, поэтому следует найти разумный компромисс между уровнем детализации планирования и организацией проекта. Обычно компромисс в таких случаях достигается за счет использования инструментальных средств организации проектов, подобных, например, Microsoft Project, во время планирования и отслеживания трудозатрат на тестирование. Отдавая предпочтение тому или иному инструментальному средству, следует убедиться, что его вход и выход совместимы с инструментальными средствами, которые применяют другие группы.

Для целей быстрого тестирования список задач должен быть точным и содержать все крупные задачи, особенно те, которые требуют времени на подготовку, например, приобретение необходимого оборудования, прием на работу квалифицированного персонала или его обучение, а также проектирование тестов. Игнорирование этапа подготовки к выполнению крупной задачи перед тестированием может привес-

Например, нет никаких оснований полагать. что каждый игполнигепь посвятит порученной плановой задаче в 40 и более часов в неделю на протяжении всего срока разработки проекта. В таком случае не останется времени на совещания, подготовку и обмен информацией. Находимо располагать временем на отладочные тесты, на решение медицинских проолгм и даже проблем личного харакгера.

Последняя позиция списка требует дополнительных пояснений. Прежде чем дополнительно подключать людеи с целью сокращения сроков решения той или иной задачи следует определить какую *задачу можно разделить на подзадачи* [40]. Например' движущиеся автомобилем может управлять только один человек, поэтому Задача управления автомобилем не допускает разбиения на подзадачи. Для задач, которые расходов.ь нэ поазадачи, н*оо*одимо «следовать две составляющих накладных расходов, подготовку исполнителей, дополнительно привлеченных к решению задачи и обмен информацией между исполнителями, которые одновременно работают над одной и той же задачей. Процесс подготовки исполнителей не допускает разбиения на части, так что трудозатраты на подготовку большего числа исполнителей возрастают в линеинои зависимости от количества привлеченных исполнителей.

Возможные пути обмена информацией между исполнителями показаны на рис 3 5 Если над задачей работает один исполнитель, то накладные расходы, обусловленные необходимостью обмена информацией, отсутствуют. Если дополнительно привлекается один исполнитель, то между двумя исполнителями добавляется двунаправленный компонент обмена информацией. Если к решению задачи привлекается третий исполнитель трудозатраты на обмен информацией возрастают втрое по сравнению со случаем участия двух исполнителей. Трудозатоаты. связанные с обменом исполнителями информацией не подчиняются линейной зависимости от количества привлеченных исполнителей но возрастают в соответствии по формуле п(п-1)/2 [40].

В общем случае, две наиболее распространенных ошибки, связанные с привлечением дополнительных рабоТНИКОВ для цокращения сроков сдачи проекта, заключаются в при пренебрежении разделения на подзадачи и

ч:е^нГбХе\ГнГоТе^нок^:особы " р е о д < ~ перечисленных выше трудностей и полу-

- Начинать следует с однозначно определенных "зафиксированных" требований.
- Основывать предположения и планы снижения рисков на результатах аналогичных проектов. выполненных ранее "
- Использовать реалистичные оценки ожидаемой экономии времени от привлечения к решению конкретных задач дополнительного персонала.

*Врезка 3.2.*

Ниже даются описания ряда распространенных технологий оценивания трудозатрат в порядке возрастания их сложности. Однако следует иметь в виду, что более сложные технологии не обязательно дают более точные оценки. Любая технология оценивания зависит от квалификации и опыта использующего ее работника и для того, чтобы получить представление о точности той или иной технологии необходимо проверить ее на статистических данных.

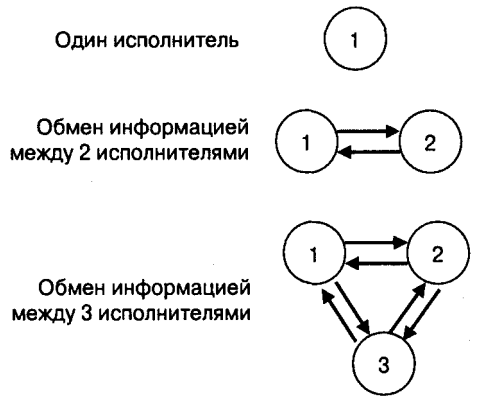



*Рис. 3.5. Обмен информацией между исполнителями*

1. **Учет ограничений, накладываемых финансовой сметой или графиком выполнения работ.** Возможны случаи, когда свобода выбора необходимого числа исполнителей отсутствует или когда заданы жесткие сроки разработки проекта. В этих условиях оценка трудозатрат также нужна, однако ее результаты будут другими. Например, может быть установлен жесткий срок проведения испытаний, однако при этом остается возможность выбора количества исполнителей этого задания. В этом случае следует рассмотреть возможность выполнения только задач с наивысшим приоритетом и одновременного выполнения максимально возможного числа задач. Однако в подобных ситуациях важно так определить ограничения и риски, связанные с тестированием, чтобы они соответствовали оценке трудозатрат.

2. **Аналогия с предыдущими проектами.** Если разрабатываемый программный продукт является очередной версией из последовательности итеративных версий или имеет много общего с некоторым завершенным программных продуктом, то в ряде случаев можно воспользоваться статистическими данным ранее разработанного проекта. При этом важно, чтобы были известны фактические издержки на разработку старого проекта, а условия разработки старого проекта должны максимально соответствовать условиям разработки нового проекта. Например, над обоими проектами должно работать одно и то же число исполнителей либо исполнители должны обладать одной и той же квалификацией.

3. **Экспертная оценка.** Количество исполнителей или сроки, необходимые для выполнения требуемых задач, подсчитывают один или большее число экспертов. Этот метод может выглядеть очень просто, когда эксперт записывает свою оценку на листике бумаги, или же очень сложно, когда требуется консенсус всех принимающих в нем участие (примером может послужить технология Wideband Delphi). Краткое описание технологии Wideband Delphi приводится во врезке 3.3.

**Методы декомпозиции.** Если программный продукт достаточно крупный и сложный, по всей вероятности, получение оценок трудозатрат на разработку и тестирования этого продукта, потребует больших затрат времени и усилий Вполне вероятно также, что в данной ситуации будет назначен руководитель проекта, ответственный за соблюдение финансовой сметы и графика выполнения работ всего проекта, а от группы тестирования потребуется предоставлять руководителю проекта исходные данные в специальном формате. Руководитель проекта может поручить каждой группе специалистов дать собственную оценку затрат с применением одной из описанных выше технологий. С другой стороны, руководитель может воспользоваться более унифицированным подходом, в рамках которого программный продукт *разбивается на блоки* либо по количеству строк кода, либо по функциональным баллам, а затем к блокам применяется некоторый оценочный алгоритм. Если используется именно такой подход, то группе тестирования целесообразно получить собственную независимую оценку, скажем, экспертную, и убедиться в том, что применение алгоритмического подхода имеет смысл.

**Модели эмпирического оценивания.** Существует множество моделей оценивания, которые могут использоваться для вычисления затрат на разработку проекта по созданию программного продукта. В основу этих моделей положено количество строк программного кода (LOC - number of lines) или функциональные баллы (FP - functional points), причем для одних и тех же исходных данных эти модели дают разные результаты. Ключевым условием внедрения любой из моделей является калибровка модели относительно локальных условий за счет ее применения на завершенных проектах и настройка ее на фактические данные так, чтобы она выдавала предсказуемые результаты. В главе 12 можно найти подробную информацию о широко распространенной технологии оценивания COCOMO.

### ПОДГОТОВКА ОЦЕНКИ ЗАТРАТ НА ТЕСТИРОВАНИЕ

Действия, которые рекомендуется выполнять при проведении оценочных работ переделены ниже Рассматриваемая процедура, представляющая собой версию процесса W.deband Delphi Estimation Process, должна осуществляться на раннем этапе процесса планирования. Этобудет гарантировать, что вас окажется достаточно времени на выполнение каждого действия.

**Подготовка.** Подготовьте формуляры для входных данных, предназначенные для сбора и документирования оценок. Подготовьте краткий пояснительный материал, объясняющий исполнителям, как проводить соответствующие работы и что от них ожидать,

Вступительное совещание. Созовите вступительное совещание для консультаций по процессу оценивания и распределите новые обязанности между членами рабочих групп чтобы они смогли разобраться с этими обязанностями и при необходимости объяснять их во время рабочих совещаний. Покидая это совещание, члены групп исполнителей должны иметь в своем распоряжении списки элементов, подлежащих оценке, и понимать, как оценка выполняется. Хорошо подумайте, прежде чем пропускать это действие - оно ставит исполнителей в равные условия и препятствует возникновению ошибок из-за нечеткого понимания самого процесса либо оцениваемых элементов.

Надомная работа. Исполнители делают индивидуальные оценки. Предоставьте участникам время, достаточное для ознакомления с новыми функциями очередной версии программного продукта.

В то же время, крайне нежелательно, чтобы кто-то из исполнителей размышлял дольше других и не мог обмениваться данными с другими. Убедитесь в том, что члены исполнительных групп понимают, что оценивают количество часов (или дней), необходимых для решения той или иной задачи при условии, что ресурсы для решения этой задачи уже имеются. Кроме того, убедитесь, что исполнители знают, как пользоваться единицами измерения (человеко-часы, человеко-дни).

**Рабочее совещание.** Созовите рабочее совещание для проверки и пересмотра оценок. Председатель совещания производит сбор оценок и заносит их в специальный сводный формуляр оценок. Сводные оценки возвращаются группе исполнителей для обсуждения и пересмотра. Группа проводит обсуждение оценок, причем каждый член группы может иметь на этот счет собственное мнение, после чего пересмотренная оценка возвращается председателю совещания. Процесс может продолжаться до тех пор, пока никто из участников не захочет менять свои оценки, либо оценки окажутся в пределах разумного компромисса, либо время, отпущенное на рабочее совещание, истечет. Совещание такого рода не может длиться дольше двух часов — если оно длится дольше, то его результаты окажутся размытыми. Во время рабочего совещания могут возникать новые проблемы, которые нельзя решить в ходе совещания. Назначьте ответственных за их решение.

**Решение проблем.** Если после рабочего совещания остаются нерешенные проблемы, примите меры по их решению либо созовите еще одно рабочее совещание, либо, если они затрагивают интересы не очень многих исполнителей, обсудите новую информацию с подгруппами, которые осведомлены в соответствующих вопросах. Скорректируйте оценки на базе новой информации.

**Получение общей оценки.** Изучите сводный формуляр оценок и отыщите для каждой задачи минимальную и максимальную оценки затрат. Возможно, на основе собственного опыта или статистических данных по другим проектам удастся отбросить резко отличающиеся значения как нереалистичные. Сложите максимальные и минимальные оценки для получения оценки трудозатрат в масштабах всего проекта. В этом случае получаются два результата: оптимистическая, или "агрессивная", оценка (минимальное значение) и пессимистическая, или "умеренная", оценка трудозатрат (максимальное значение). Задокументируйте все предложения, которые были сделаны в процессе получения окончательной оценки.

**Преобразование оценки в график работ.** Теперь, когда имеется оценка требуемых трудозатрат, настала очередь ответа на вопрос, на который вы обязаны ответить. Когда будет закончена работа? Одно из эмпирических правил ([33], с. 183) выглядит так:

$$\textit{Продолжительность графика работ в месяцах} = 3{,}0 * \text{человеко-часы}^{1/3}$$

Разумеется, имеет смысл получить более точные сроки, нежели те, что вычисляются по этой незамысловатой формуле.

*Врезка 3.3.*

## Определение продолжительности задачи и построение графика работ

После того, как задачи определены, а трудозатраты на каждую задачу подсчитаны, потребуется определить время решения каждой задачи и исследовать первый разрез графика работ. Время, необходимое для выполнения задачи, называемое также продолжительностью, зависит от количества исполнителей, могущих трудиться над ней. Если есть возможность привлечь к выполнению той или иной задачи более одного исполнителя, убедитесь сначала, что эта задача допускает разбиение на подзадачи — другими словами, выполнением этой задачи могут одновременно заниматься более одного исполнителя. Некоторые работы, подобные жарке бифштекса, не будут вы-

полнены быстрее, если к ним вместо одного привлекаются несколько человек. Более подробный анализ этого вопроса можно найти во врезке 3.2.

После назначения на каждую задачу конкретных исполнителей следовало бы выяснить, какие задачи могут выполняться одновременно, а какие — только в определенной последовательности. Одним из факторов, которые должны исследоваться при составлении плана выполнения задач, является использование одного и того же оборудования. В вашем распоряжении может оказаться достаточно исполнителей, готовых выполнять работы одновременно, но вы должны также принять во внимание, насколько доступным будет оборудование. При наличии достаточного количества единиц совместно используемого оборудования, возможно, придется построить график использования этого оборудования или базу данных обслуживания, которая будет контролировать работу исполнителей, степень использования оборудования и время тестирования. Вопрос создания баз данных обслуживания подробно рассматривается в [33].

Задачи, выделенные ресурсы (время и оборудование) и продолжительность выполнения можно графически отобразить на гистограмме, пример которой показан на рис. 3.6. Такая гистограмма может использоваться для отображения ключевых этапов проекта, связанных со "знаменательными" событиями в жизни проекта, как то: начало тестирования, конец тестирования или дата поставки программного продукта. На рис. 3.6 эти этапы помечены значком +. На диаграмме, представленной на рис 3.6, приводятся только названия задач и связанные с ними даты. На более подробной гистограмме могут быть показаны ресурсы, выделенные для задачи, тексты, указывающие начальные и конечные даты, а также другая информация, имеющая отношение к управлению выполнением проекта.

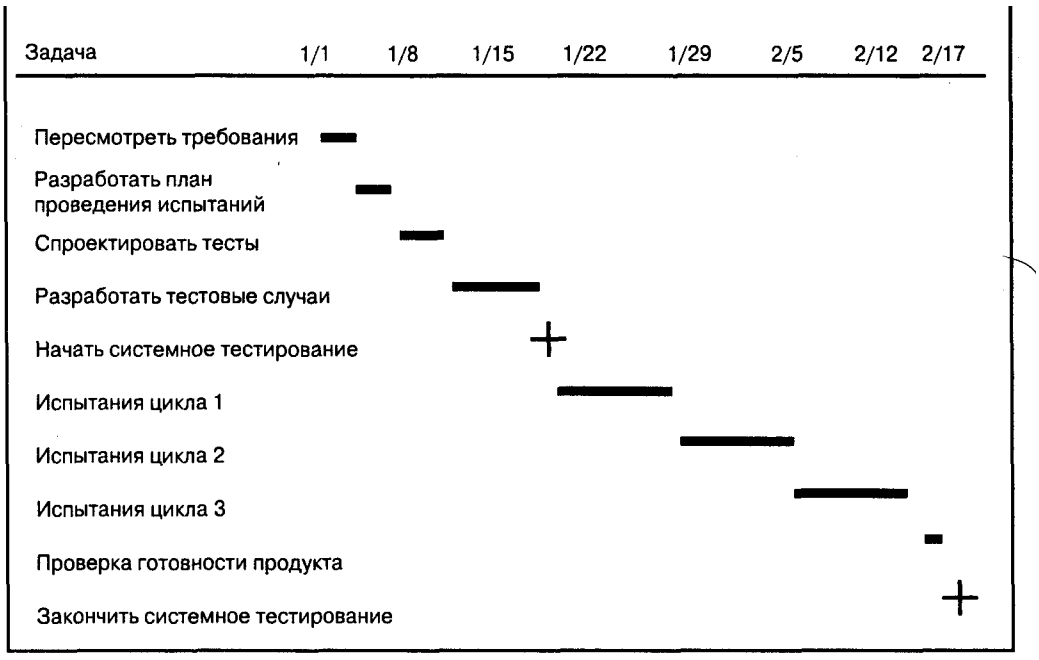



*Рис. 3.6. Пример гистограммы*

Часто знаменательные даты проекта сводятся в отдельную таблицу, которая получила название поэтапного графика. Примером поэтапного графика, который может использоваться руководством и персоналом, проводящим испытания, служит таблица 3.4.

**Таблица 3.4. Пример поэтапного графика**

| Этап | Дата |
|---|---|
| Завершение разработки плана тестирования | 1/8 |
| Завершение разработки тестового случая | 1/18 |
| Начало системных испытаний | 1/18 |
| Цикл 1 испытаний завершен | 1/25 |
| Цикл 2 испытаний завершен | 2/1 |
| Цикл 3 испытаний завершен | 2/12 |
| Проверка готовности программного продукта | 2/14 |
| Завершение системных испытаний | 2/14 |

## Оценка рисков, связанных с графиком работ

Как только будет получена оценка количества исполнителей, оборудования и времени, необходимого для тестирования программного продукта, потребуется оценить риски, связанные с этими оценками. Это делается исходя из предположения, что может выйти из строя во время проведения испытаний. Далее несложно составить план устранения возникающих проблем. Объем трудозатрат, необходимых для смягчения последствий от овеществленных рисков, даст представление о том, насколько полученные оценки могут отклоняться от фактических значений затрат времени и средств. Риски, встроенные в полученные оценки, — это первое, о чем следует доложить вышестоящему руководству в процессе пересмотра оценок.

Ниже приводятся примеры рисков, которые оказывают слияние на график работ по тестированию:

- Аппаратные средства, необходимые для проведения тестирования, отсутствуют на начальной стадии испытаний
- Тестируемый программный продукт не поступил на испытания
- Тестовые случаи не готовы к началу испытаний
- Исполнители, которым поручено проведение испытаний, не могут приступить к тестированию
- Внесение изменений в требования в процессе разработки тестов или во время испытаний
- Изменение пользовательских интерфейсов в процессе разработки тестов или во время испытаний
- Освоение персоналом новых средств тестирования не завершено к началу испытаний

Анализ рисков часто влечет за собой разработку плана смягчения последствий от овеществления рисков. Например, если вы убеждаетесь в том, что тестовые случаи не

готовы ко времени проведения испытаний, появляются все основания предложить привлечь к испытаниям дополнительные трудовые ресурсы. Если работы по тестированию зависят от поставки нового оборудования, необходимого для реализации проекта, то, заложив эту зависимость в график работ и давая оценку связанных с ней рисков, вы ставите в известность руководство проекта о том, что сроком поставки нового оборудования должно быть начало испытаний, а не дата его отгрузки.

Обычная практика управления при допущении риска предусматривает связь каждого риска с вероятностью осуществления этого риска и меры воздействия на проект, если риск все же случится. Например, возможно, вскроется тот факт, что, скорее всего (с вероятностью 90%), основные специалисты по тем или иным причинам не смогут приступить к тестированию в запланированные сроки, поэтому возможность нарушения графика работ оценивается как критическая (по шкале критическая, высокая, средняя, малая). Риск, который оказывает большое влияние на выполнение проекта и характеризуется высокой вероятностью осуществления, относится к категории тех рисков, в случае возникновения которых у вас должен быть план дополнительных мероприятий. В подобной ситуации, возможно, потребуется обсудить возможность внесения изменений в план работ или изменить очередность подключения специалистов различного профиля, участие которых необходимо для проведения испытаний.

Предполагаемый перечень действий для оценки рисков выглядит следующим образом:

- Составить список всех рисков, какие, по вашему мнению, представляют угрозу выполнению графика работ по тестированию. Проведите анализ статистических данных, таких как оценки выполнения проектов или информация о необычных случаях сбоев, имевших место при выполнении предыдущих проектов. Получите оценку зависимости от других групп, обсудив с ними вероятность того, что они выполнять предложенные вами критерии вхождения в испытания.
- Дать оценку вероятности осуществления каждого выявленного риска.
- Дать оценку последствий осуществления риска в отношении выполнимости плана испытаний. Для описания этого влияния воспользуйтесь шкалой критическое, высокое, среднее, малое.
- Разработать план снижения вероятности осуществления рисков или их влияния, начиная с тех из них, которые обладают высокой вероятностью осуществления или чреваты тяжелыми последствиями.

Существует много литературы по вопросам управления при допущении риска. Для пополнения своиъ знаний рекомендуем обратиться к [43], [42] и [30], которые содержат краткую, но в то же время весьма полезную информацию.

# Подготовка и пересмотр документов, содержащих планы проведения испытаний

## Формат плана проведения испытаний

После выбора стратегии тестирования, определения испытательной системы и оценки трудозатрат можно приступать к составлению плана проведения испытаний. План проведения испытаний представляет собой документ или некоторый набор документов, представляющих собой результат выполнения определенных видов деятельности по планированию испытаний. Этот документ не является раз и навсегда установленным планом, он меняется по мере изменения требований. Полезные рекомендации по разработке плана проведения испытаний предлагает стандарт IEEE Standard 829: *IEEE Standard for Software Test Documentations* [22] (стандарт IEEE по составлению документированию испытаний программного обеспечения). Конечно, можно воспользоваться другим форматом, однако потребуется оформить все материалы, рекомендуемые указанным выше стандартом IEEE в любом подходящем формате. Другие идеи и форматы рассматриваются в [15], а также в [30], [40] и [33]. В данном разделе при построении шаблона плана проведения испытаний будет использоваться стандарт IEEE Standard 829.

Согласно рассматриваемому стандарту, план проведения испытаний должен содержать 16 разделов:

1. Идентификатор плана проведения испытаний
2. Введение
3. Компоненты, которые должны тестироваться
4. Характеристики и свойства, которые должны тестироваться
5. Характеристики и свойства, которые не должны тестироваться
6. Подход
7. Критерий успешных и неудачных испытаний
8. ' Критерий приостановки испытаний и требования возобновления испытаний
9. Выходные результаты тестов
10. Задачи тестирования
11. Требования окружающей среды
12. Распределение ответственности
13. Подбор кадров и подготовка персонала
14. График работ
15. Риски и непредвиденные обстоятельства
16. Утверждение плана проведения испытаний.

Если внимательно проанализировать этот список, несложно понять, чем этот план *не является* — он не представляет собой подробное описание того, как должен выполняться прогон тестов, и не является местом сосредоточения результатов тес-

тирования. Некоторые организации, специализирующиеся на тестировании программных продуктов, объединяют один или большее число пунктов списка с содержимым плана проведения испытаний, тем самым организуя всю тестовую документацию в одном месте. Это может дать прекрасные результаты, однако в данной книге мы преследуем цель достижения большей модульности комплекта документов, которая совпадает с целью упомянутого выше стандарта.

Тот факт, что указанный стандарт направлен на разделение планов проведения испытаний, процедур и результатов, становится очевидным из содержащихся в нем определений следующих документов [22].

**План проведения испытаний.** Это документ описывает возможности, подход, ресурсы и график исполнения различных видов тестовой деятельности. Он определяет объекты тестирования, функции, которые должны тестироваться, тестовые задания, исполнителей каждого тестового задания, и любой из рисков, требующий планирования на случай непредвиденных обстоятельств.

**Спецификация методики тестирования.** Этот документ описывает последовательность действий при выполнении конкретного теста.

**Отчетный доклад о тестировании.** Этот документ обобщает виды тестовой деятельности и их результаты. Он также содержит оценку соответствующих тестовых объектов.

Спецификация методики тестирования изучается в главе 4 как часть работ, выполняемых при разработке тестов, а отчетные доклады о тестировании рассматриваются в главе 5, в которой речь идет о выполнении тестов. В идеальном случае все эти документы должны быть связаны в эффективную, скоординированную систему, которая поддерживает процесс тестирования от начала до конца.

В данном разделе мы проведем анализ содержимого плана проведения испытаний и оптимизацию его структуры для целей быстрого тестирования. По каждому пункту этого плана мы попытаемся найти приемлемый способ быстро и эффективно удовлетворить его требования. Разумеется, некоторые предложенные решения не будут соответствовать вашим потребностям, но тогда вы должны будете выполнить такой же анализ в контексте собственной среды разработки.

Каждый пронумерованный ниже пункт соответствует пункту эскиза плана, приведенного в предыдущем разделе. Примеры плана проведения испытаний, спецификации методики тестирования и отчета о тестировании приводятся в третьей части книги.

**1. Идентификатор плана проведения испытаний.** В долгосрочной перспективе можно сэкономить время, если присвоить всем тестовым документам идентификаторы, которые нетрудно отслеживать. По существу, возможность отслеживания документов позволяет получить следующие преимущества:

- Наличие уникального идентификатора каждого тестового документа позволяет воспрепятствовать тому, что кто-либо из исполнителей напрасно затратит время на изучение не того документа, что надо. Таким идентификатором может быть алфавитно-цифровая строка, которая может содержать данные, отображающие название продукта, номер версии, дату и любую другую информацию, позволяющую отслеживать этот документ. Рекомендуется отдавать предпочте-

ние идентификаторам, которые вписываются в общую систему управления документов.

- Построить хранилище документов (возможно, сетевой дисковый накопитель), которое позволяет исполнителям и руководству быстро находить документы, имеющие отношение к тестированию. Если во время очередного кризиса вам придется отыскивать нужную версию того или иного документа, вы убедитесь, насколько полезным является подобное хранилище документов.
- Построить систему отслеживания документов, поддерживающую управление версиями. В любой момент может возникнуть необходимость внести изменения в тестовые документы, в том числе планы проведения тестирования. Вы должны иметь возможность донести до исполнителей и руководства последние изменения. Управление версиями может быть таким же простым, как документирование статистических данных на титульном листе документа и поддержание актуальной версии, помещенной в хранилище документов.

**2. Введение.** Цель введения заключается в том, чтобы дать краткую информацию для любого исполнителя, желающего воспользоваться планом проведения испытаний. Суть состоит в необходимости указания во вводном разделе плана проведения испытаний исходных документов, служащих входными данными выполняемого тестирования.

Например, можно составить список документов с требованиями и функциональных спецификаций для тестируемого программного продукта. При составлении этого списка желательно указать расположения документов (путь в сетевом дисковом накопителе или URL-адрес Web-страницы), что даст возможность быстро получить к ним доступ. Наиболее подходящий способ заключается в помещении в план проведения испытаний ссылки на исходный документ, благодаря которой пользователь сможет переходить из недокументальной копии плана на соответствующие справочные материалы. Преимущество такого подхода связано с возможностью автоматического выхода пользователя на последнюю версию входного документа.

Другим аспектом, который совершенно не связан с обеспечением быстрого поиска входных документов, является минимизация временных затрат на написание введения. Пример, сопровождающий стандарт IEEE Standard 829, содержит цели плана проведения испытаний, предварительные данные о программном продукте, объем тестирования, предусмотренный планом проведения испытаний, и перечень ссылок. Размер такого документа не превышает половины страницы. Если при этом имеются ограничения, оказывающие влияние на процесс планирования, например, срок выхода продукта на рынок, их можно перечислить здесь же.

**3. Компоненты, которые должны тестироваться.** Здесь приводятся высокоуровневые списки компонент, которые вы намерены тестировать. В этом разделе будут упомянуты некоторые из них:

- Идентификатор выпуска/версии тестируемого программного проекта
- Исправления дефектов, обнаруженных в предыдущей версии. Если список дефектов приводится в спецификации функций, необходимо указать ссылку на спецификации функций и не дублировать их здесь

- Описание среды, используемой для распределения программного продукта (например, компакт-диски, Web- или ftp-сайты)
- Документы для конечного пользователя, такие как руководство пользователя, инструкции по установке, примечания по версии продукта.

Обратите внимание на тот факт, что это перечень представляет собой ни что иное, как входные данные для выполняемого тестирования. Все эти материалы должны быть получены своевременно, чтобы не нарушать сроки тестирования, поэтому в плане они помечаются как зависимости. Следует обладать полной ясностью относительно того, что тестируется, дабы ничто не смогло ускользнуть из внимания. Например, существует вредная привычка до последней минуты "забывать" проводить тестирование документов для пользователя. Пытаясь исправить подобную забывчивость в конце разработки проекта, легко допустить срыв графика работ и, что еще хуже, поставить заказчику продукт с необнаруженными дефектами.

**4. Характеристики и свойства, которые должны тестироваться.** Стандарт IEEE Standard 829 предлагает следующее определение *свойства программного обеспечения (software feature):* это отличительная характеристика программного компонента, например, производительность, переносимость или функциональные возможности.

Это очень общее определение, поскольку оно должно охватить большое количество понятий. Возможно, имеет смысл давать определения свойств в бизнес-терминах. Рассмотрим свойство быть тем, "что может продаваться заказчику". Например, если приложение работает медленнее, чем это нужно заказчику, то увеличение быстродействия его наиболее часто используемых функций может подаваться как свойство. Более совершенный новый способ ввода ускорения ввода данных или запроса в базу данных также может продать как свойство.

Ключевым моментом для специалистов по тестированию является то, что если что-то было обещано заказчику, оно должно быть отражено в плане проведения испытаний с тем, чтобы протестировать его до поставки объекта заказчику. Основным источником, позволяющим узнать, что было обещано заказчику, служат документы определения требований, на которые существуют ссылки во введении.

Перечень свойств в этом разделе предоставляет в ваше распоряжение контрольную таблицу для вычисления трудозатрат, необходимых для проведения испытаний программного продукта. Каждая позиция этого списка соответствует конкретным пунктам документов описания требований, в то же время каждой такой позиции должен соответствовать один или большее число тестов, определенных в спецификации методики тестирования.

**5. Характеристики и свойства, которые не должны тестироваться.** Этот раздел предназначен для того, чтобы заблаговременно и однозначно определить, какие объекты не должны охватываться тестированием. В число объектов, объявляемых как "не подлежащие тестированию", могут входить следующие:

- Функции, реализация которых отложена до последующих инкрементальных версий продукта

- Конфигурации компьютерных средств, которые невозможно проверить по причине отсутствия необходимого оборудования. Иногда используются настолько дорогостоящие конфигурации компьютерного оборудования, что их невозможно продублировать в условиях испытательных лабораторий. Если вы не можете воспроизвести операционную среду, но в то же время можете ее смоделировать, об этом потребуется сообщить в разделе "Подход". Если вы вообще не можете проводить испытания в операционной среде, этот вопрос следует рассмотреть в разделе, в котором обсуждаются "риски".
- Комбинации настроек или конфигурации аппаратных средств, которые не могут быть испытаны за время, выделенное для выхода программного продукта на рынок. Если принято решение ограничить объем тестирования некоторого конкретного свойства, необходимо обсудить данные ограничения и обосновать их в этом разделе.

На завершающих стадиях проекта, когда, как правило, на группу тестирования оказывают давление в плане поторопиться с окончанием работ, очень полезно иметь в своем распоряжении документ, в котором оговаривается, что планируется тестировать, а что нет. В пылу предфинишной лихорадки люди склонны забывать, какие соглашения были достигнуты — совсем неплохо опереться на этот раздел плана проведения испытаний при ответах на вопросы, почему то или иное свойство не охвачено тестированием. Разумеется, если проблема возникнет в области, которая не подвергалась тестированию в вашей лаборатории, она должна быть устранена и проверена после выпуска очередной версии программного продукта. Поскольку стоимость исправления дефектов после сдачи программного продукта в эксплуатации очень велика, важно получить по возможности точную оценку рисков, прежде чем принимать решение не проводить испытаний того или иного свойства или конфигурации.

в. Подход. Этот раздел плана проведения испытаний предназначен для описания на высоком уровне, как вы намерены проводить испытания программного продукта. Это описание не является подробной спецификацией всех методик испытаний, которые планируется использовать. Подход, описание которого включено в план проведения испытаний, должен быть основано на соображениях, рассмотренных в одном из предыдущих разделов, а именно, в разделе "Определение подхода к тестированию". В этот раздел можно включить, например, такие темы:

- Статическое тестирование требований и проектная документация
- Статическое и динамическое тестирование, которое должно проводиться на стадиях тестирования программных кодов, модулей и проверки взаимодействия и функционирования компонентов системы
- Тестирование свойств
- Испытания при перегрузках/испытания под нагрузкой/тестирование производительности
- Проверка средств защиты
- Тестирование установки/обновления программного продукта
- Тестирование средств дублирования/восстановления
- Тестирование GUI-интерфейса

- Регрессивное тестирование
- Приемочные испытания: альфа-, бета- и другие виды испытаний на месте
- Проверка результатов устранения дефектов
- Прерывание испытаний, проводимых в автоматизированном и неавтоматизированном режимах
- Виды тестирования, проведение которых поручается сторонним организациям
- Использование системы отслеживания дефектов для ввода сообщений о неисправностях.

Если реализуется лишь некоторая часть той или иной долгосрочной стратегии, например, автоматизация процесса тестирования и тестовых случаев, можно определить, какие конкретные аспекты соответствующего долгосрочного плана будут реализованы в текущем проекте проведения испытаний, и сослаться на документ, в котором описывается стратегия автоматизации. Например, возможно, планируется применение некоторого инструментального средства или впервые предпринимается попытка автоматизации тестирования GUI-интерфейса. Кроме того, может планироваться использование услуг сторонних тестовых организаций для проведения испытаний продукта под нагрузкой и в условиях перегрузки. Этот раздел представляет собой подходящее место для описания плана на концептуальном уровне и ссылок на любые письменные соглашения, которые удалось заключить со сторонними организациями.

При выполнении тестирования инкрементальных версий программного продукта подход к тестированию каждой версии, по всей видимости, будет одним и тем же. Вы должны обладать возможностью экономить время за счет заимствования больших фрагментов того же раздела плана проведения испытаний, составленного для предыдущих версий, но в то же время тщательно анализировать последствия любых изменений, внесенных в текущую версию программного продукта.

**7. Критерии успешного/неудачного прохождения испытаний.** Два критерия успешного/неудачного прохождения испытаний заслуживают особого внимания при тестировании. Первый относится к индивидуальным тестам. Все тестовые случаи должны снабжаться наборами ожидаемых результатов; по существу, если получены ожидаемые результаты, итог прогона теста считается успешным, если нет — результат прогона теста расценивается как неудачный. Критерии успешного/неудачного прохождения отдельных тестов должны быть включены в сами тестовые случаи, в связи с чем их не следует помещать в план проведения испытаний. Однако не лишним будет указать в этом плане, что определение критериев успешного/неудачного прохождения испытаний является неотъемлемой частью тестового случая.

Второй тип критериев успешного/неудачного прохождения испытаний имеет отношение к тестированию всего программного продукта. С самого начала вы должны определить, что следует считать успешным прохождением стадии тестирования, т.е. когда можно прекратить испытания продукта и наладить его поставку. В некоторых случаях критерий выхода из испытаний определяется в программе испытаний или даже в документе, который содержит формулировки требований, предъявляемых к программному продукту. Вне зависимости от того, что служит источником этих критериев, полезно дать их четкую формулировку в плане проведения испытаний.

Разумеется, нереально ожидать, что любые критерии, объявленные в плане проведения испытаний, будут автоматически управлять выпуском программного продукта. Выпуск программного продукта определяется соображениями экономического характера, которые основываются на информации о качестве продукта, но в то же время во внимание принимаются и ряд других экономических факторов.

**8. Критерий приостановки испытаний и требования возобновления испытаний.** Выше в этой главе мы установили, что критерий вхождения в испытание показывает, что необходимо предпринять, прежде чем начнете тестирование, а критерий выхода из испытаний описывает, что, по вашему мнению, требуется для завершения испытаний. Критерии приостановки и возобновления испытаний описывают, что происходит, когда продолжению испытаний препятствуют дефекты. Например, можно объявить, что если модуль содержит устойчивую системную ошибку, исключающую успешную установку программного продукта, тестирование приостанавливается на период времени, необходимый для устранения этой ошибки. Разумеется, нельзя останавливать тестирование каждый раз, когда какая-то часть тестов заблокирована дефектами, однако следует поставить в известность руководство в ситуациях, когда программный продукт содержит такое количество дефектов, что продолжать испытания нецелесообразно.

**9. Выходные результаты тестов.** Этот раздел плана проведения испытаний является тем местом, в котором определяются выходные данные работ по тестированию, например, можно предложить следующий список:

- План проведения испытаний
- Документы, регламентирующие проектирование тестов
- Документ, содержащий спецификацию тестов
- Сообщения о результатах прогона тестов
- Сообщения об обнаруженных дефектах
- Примечания по выпуску программного продукта (если это вменяется вам в обязанности).

Всегда полезно напомнить ответственным исполнителям конкретных документов .предельные сроки оформления документа. Чтобы исключить напрасную трату времени на более поздних стадиях, можно подготовить шаблоны, или "козы", этих документов на этапе составления плана проведения испытаний, а затем установить связи или ссылки на "фиктивные" документы. Например, может оказаться полезным заранее подготовить шаблон отчета по результатам тестирования и заполнять его тестовыми данными по мере их готовности. Кроме того, шаблон проекта теста может ускорить работы по проектированию тестов. Удивительно, как много времени можно понапрасну терять, споря о формате и содержании любого из перечисленных выше документов.

Определение и согласование того, что вы намереваетесь опубликовать по завершении тестирования на раннем этапе жизненного цикла, может стать хорошей практикой, поскольку в этом случае упрощается построение реалистичного плана работ. Дабы не закладывать в план работ неоправданных накладных расходов, нежелатель-

но отображать в плане результаты работ, которые не существенны для достижения конечной цели, каковой в рассматриваемом случае является максимально быстрая поставка программного продукта.

**10. Задачи тестирования.** Если план проведения испытаний применяется для обоснования произведенной оценки трудозатрат на тестирование, этот раздел является подходящим местом для определения индивидуальных задач, которые необходимы для подготовки и выполнения тестирования. Еще лучше, если вы занесете вашу оценку в динамическую электронную таблицу. Другой способ предполагает использование программы управления проектами, например, Microsoft Project, для того, чтобы в этом разделе плана организовать ссылку на другой документ. Это еще один подходящий момент, чтобы указать ссылку на справочный документ или, по крайней мере, путь к соответствующему файлу или сетевому дисковому накопителю.

**11. Информация о конфигурации средств тестирования (требования окружающей среды).** Дайте описание аппаратных и программных средств, необходимых для проведения испытаний, и начертите диаграммы, показывающие, как аппаратные компоненты соединяются друг с другом. Этот раздел предназначен для того, чтобы позволить любому другому заинтересованному лицу построить такую же испытательную установку в случае необходимости воспроизведения неисправности или проверки, что неисправность устранена. Данный раздел плана проведения испытаний должен составляться на базе анализа, проведенного в разделах "Среда тестирования" и "Конфигурации средств тестирования" ранее в этой главе. Возможно, потребуется расширить рамки этого раздела и включить в него архитектуру тестов (т.е. как организованы тесты) и инструментальные средства тестирования, которые вы намерены использовать. Обе темы затрагиваются в разделе "Определение испытательной системы" в начале главы.

**12. Распределение ответственности при проведении тестирования.** Если возможности вашей тестовой группы по завершению тестирования каким-то образом зависят от деятельности другой группы, в этом разделе плана проведения тестирования должно быть указано, кто что делает. Например, если вы рассчитываете на разработчиков программного обеспечения в том, что в проводимую ими проверку взаимодействия и функционирования компонентов программного продукта будет включен некоторый конкретный набор тестов, данный раздел является подходящим местом для формулировки соответствующих условий. Возможно, в этом испытании намерены принять участие инженеры, обслуживающие заказчика, занявшись тестированием пользовательского интерфейса или проверкой документации.

В качестве другого примера распределения ответственности, о чем целесообразно упомянуть в этом разделе, можно привести привлечение к тестовым работам территориально удаленных организаций (например, привлеченная к сотрудничеству по контракту лаборатория в другом городе или даже в другой стране), которые принимают на себя выполнение определенной части тестовых работ. Во всех этих случаях полезно составить матрицу ответственности, где задачи соотносятся с исполнителями.

Если ни одна из упомянутых ситуаций не имеет места, в этом разделе можно поместить стандартную фразу, дескать, организация XYZ, специализирующаяся на тестировании, будет проводить испытания программного продукта и сообщать об обнаруженных дефектах разработчикам. В этом разделе не указываются служебные обязанности конкретных инженеров и техников; для этой цели предназначен следующий раздел плана проведения испытаний.

**13. Подбор кадров и подготовка персонала.** В этом разделе можно привести список персонала, который вы намерены привлечь к тестированию программного продукта. Одна из причин включения такого списка связана с тем, что своевременное выполнение ключевых этапов тестирования зависит от возможности участия этих людей на стадии тестирования. Если кто-либо из этих специалистов был переброшен на другой проект или вообще уволился, у вас появляются серьезные основания для привлечения других исполнителей или смещения сроков тестирования.

Этот раздел также является подходящим местом для определения, какую подготовку должен пройти персонал группы тестирования, чтобы приобрести квалификацию, которая потребуется для проведения запланированных испытаний.

**14. График работ.** Этот раздел обычно не является местом, в котором оглашается подробный график проектных работ — для таких целей существует общий план проектных работ или самостоятельный файл, который может подвергаться изменениям независимо от плана проведения испытаний. В конце концов, подробный план работ может быть очень динамичным документом, однако содержимое того, что вы тестируете, кадровые проблемы, рабочие продукты тестирования и другие компоненты плана проведения испытаний остаются достаточно статичными.

Скорее всего, в план проведения испытаний потребуется включить краткий список ключевых этапов тестирования, например, срок начала тестовых работ, когда бета-версия продукта должна поступить к первым пользователям и когда завершатся испытания. С другой стороны, можно просто принять решение обращаться к планам проектных работ или самостоятельному плану работ, используя для этих целей ссылку на Web-странице или путь к сетевому дисковому накопителю.

**15. Риски и непредвиденные обстоятельства.** Риски могут описываться в таблице, в которой указываются риски, вероятность овеществления этого риска, влияние этого **риска** и план мероприятий по снижению от овеществления рисков. Процесс построения такой таблицы описан в разделе "Оценка рисков невыполнения графика **работ".**

**16. Утверждение плана проведения испытаний.** Для решения этой задачи на титульном листе плана проведения тестовых работ потребуется составить список лиц, утверждающих план, и получить их одобрение. Можно также отправить утверждающему лицу по электронной почте копию плана проведения испытаний и получить **ответ** с утвержденным планом. Хранить сообщения и ответы электронной почты рекомендуется в совместно используемом сетевом каталоге, который дублируется через регулярные промежутки времени, либо присоединить все ответы и подтверждения к специальному приложению к плану проведения испытаний.

## Проверка выполнения плана проведения испытаний

План проведения испытаний — это важный рабочий продукт жизненного цикла разработки, в силу чего он должен быть подвергнут тщательной проверке. По возможности он должен пройти формальную проверку (см. раздел "Методы статического тестирования" в главе 2 и раздел "Проверки/сквозной контроль/экспертные оценки" в главе 9, где содержится более подробная информация о проверках). В проверках должны участвовать не только члены группы тестирования, но и представители коллективов разработчиков, групп маркетинга и руководства (по меньшей мере, через электронную почту). Проверки преследуют сразу несколько целей:

- Все исполнители, принимающие участие в разработке программного продукта, должны понимать и одобрять цели, покрытия, ограничения и риски, связанные с проведением тестирования.
- Подход к тестированию должен быть пересмотрен с целью обеспечения его технической корректности и обеспечения проверки им всех требований, выполнение которых было обещано заказчику.
- Критерии входа и выхода из испытаний должны правильно пониматься всеми исполнителями, принимающими участие в проекте. Любые зависимости от деятельности других групп должны быть четко зафиксированы, а группы, ответственные за те или иные этапы разработки, в установленные сроки должны предоставлять группе тестирования соответствующие рабочие продукты.

Несмотря на то что обмен информацией между группой тестирования и другими группами должен поддерживаться в течение всего процесса разработки программного продукта, периодические проверки хода выполнения плана проведения испытаний представляют собой прекрасную возможность обеспечить согласованное выполнение проектных работ на ранних стадиях жизненного цикла. Любые недоразумения и ошибки при выборе тестового покрытия, которые останутся незамеченными при проверке хода выполнения плана проведения испытаний, могут серьезно помешать проведению испытаний в запланированные сроки.

# Что дальше

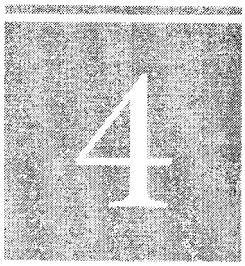
В этой главе мы обсудили планирование работ по тестированию и получение оценок. Затраты времени и усилий на планирования могут серьезно повлиять на способность вашей группы успешно реализовать план тестирования и достичь его целей. Ключевыми этапами процесса составления плана тестирования являются:

- Определение стратегии тестирования
- Определение состава и структуры испытательной системы (аппаратные и программные средства)
- Оценка трудозатрат на проведение тестирования (необходимые ресурсы и график работ)
- Оценка рисков и выполнения графика работ и подготовка планов смягчения последствий от овеществления рисков
- Подготовка и проверка документов плана проведения испытаний.

Выходные результаты этих этапов представляют собой набор документов, определяющих план проведения испытаний, который используются при управлении дальнейшими испытаниями.

В следующей главе мы познакомимся, с чем придется столкнуться во время разработки тестов. Стратегия тестирования, архитектура тестов и конфигурации средств тестирования, определения которых были даны на стадии планирования, играют важную роль при разработке набора эффективных тестовых случаев.

# Проектирование и разработка тестов



## Темы, рассматриваемые в главе:

- Разработка тестов
- Разработка тестовых случаев
- Пересмотр и отладка тестов

Q Автоматизация тестовых случаев

Q Что дальше

Ключ к успешным испытаниям лежит в эффективности выбранных тестовых случаев. Как было показано в главе 3, четкое планирование и подготовка тестирования играют большую роль, но в условиях проведения окончательного анализа выбранные случаи тестирования должны обеспечить обнаружение дефектов в программном продукте, иначе все планирование и подготовка окажутся напрасными. Цель данной главы заключается в том, чтобы подготовить базу для проектирования и разработки надежных динамических тестовых случаев, которые могут использоваться для системных и приемочных испытаний.

Диаграмма действий, обеспечивающих проектирование и разработку тестов, показана на рис. 4.1. Основными входными данными для этого процесса является набор документов, составляющих план проведения испытаний, которые были описаны в главе 3. План проведения испытаний должен дать описание подхода, который предусматривается задействовать при проведении тестирования, а также объемы трудозатрат на тестирование. В нем должна быть определена архитектура тестов, т.е., по меньшей мере, должны быть определены соответствующие наборы тестов. В нем также должен быть определен набор конфигураций средств тестирования, вокруг которых проектируются тесты.

Выходными результатами таких видов деятельности, как проектирование и разработка, являются набор пересмотренных и отлаженных тестовых случаев, которые готовы для использования в системных и приемочных испытаниях. Должно быть обеспечено обратное отображение тестовых случаев на технические требования. Кроме того, тестовые случаи должны обеспечить хорошее покрытие тестами, по меньшей мере, всех технических требований с наивысшими приоритетами, а в идеальном случае — абсолютно всех требований заказчика. Тестовые случаи должны

обеспечить хорошее покрытие программного кода продукта за счет выполнения большей части, если не всех, логических путей в программном коде. По мере возможностей, существенная часть тестовых случаев должна быть автоматизирована для поддержки высококачественно регрессивного тестирования.


*Рис. 4.1. Проектирование и реализация тестов*

Как показано на рис. 4.1, с разработкой тестовых случаев связан собственный жизненный цикл. Цикл предусматривает стадию проектирования, на которой формулируются цели теста, спецификации входных данных, определяются конфигурации средств тестирования. Цикл включает также этап разработки, в рамках которого даются подробные определения методик тестирования, а также этап проверки и отладки, на котором тесты подвергаются пересмотру и отладке. Каждый этап жизненного цикла разработки конкретного тестового случая рассматривается далее в этой главе. Более подробная информация о технологиях, используемых при динамическом тестировании, приводится во второй части, а примеры тестовых случаев можно отыскать в третьей части книги.

# Разработка тестов

Как отмечалось в главе 3, подготовка к тестированию подобна разделению луковицы на чешуйки — это многоуровневый процесс последовательной конкретизации условий. Подготовка начинается с концептуального определения стратегии тестирования, затем к нему добавляются все большее количество слоев детализации, которые описывают архитектуру тестирования и условия испытаний. В конечном итоге будут

разработаны проверенные и отлаженные методики тестирования, собрана испытательная система, после чего можно смело приступать к системным испытаниям.

Разработку тестовых случаев можно сравнить со снятием одного слоя луковицы. Проекты тестов содержат в себе больше подробностей, нежели концептуальный план построения тестов, но в то же время они еще не предусматривают конкретных действий, необходимых для прогона конкретного теста. Многоуровневый подход к разработке тестов подобен подходу, используемому при разработке программного обеспечения. В рамках высокотехнологичного процесса разработчики не переходят от формулирования требований непосредственно к написанию программных кодов. Сначала они выполняют предварительный или системный проект, в котором определяют концепцию программного продукта, после чего составляют программу или рабочий план, в котором оговариваются все детали проектирования. В области тестирования план проведения испытаний играет такую же роль, что и проектное задание в области разработки программного обеспечения, а проект теста соответствует рабочему проекту программного обеспечения.

Одним из признаков высокого качества технического проектирования является его *модульность.* В условиях модульного проектирования система разбивается на отдельные компоненты, при этом каждый компонент имеет свое назначение и четко определенные входы и выходы. Принципы модульного проектирования часто применяются при проектировании программного обеспечения; они так же хорошо подходят и для проектирования тестов. Модульным компонентом в тестировании является тестовый случай. Это означает, что перед каждым тестовым случаем должна быть поставлена четко сформулированная цель, чтобы было ясно, что подвергается тестированию. Для каждого тестового случая должна быть строго определена среда тестирования с известными начальными условиями, благодаря чему можно рассчитывать на то, что при каждом прогоне конкретный тест будет выдавать одни и те же результаты. Наконец, каждый тестовый случай должен давать строго определенный ожидаемый результат (выход) с тем, чтобы можно было воспользоваться однозначным критерием успешного/неудачного испытания.

Другим свойством модульного проектирования является то, что модули организуются в иерархию. Иерархия есть результат разбиения системы на компоненты, и она позволяет в каждый конкретный момент времени работать с одним уровнем системы. Это обстоятельство отражается на архитектуре тестов, которая может включать в себя несколько тестовых наборов, ориентированных на проверку высокоуровневых функциональных свойств системы, а также тестовые наборы или тестовых случаи, служащих для проверки детализированных функциональных свойств системы. Например, может быть один тестовый набор, предназначенный для проверки GUI-интерфейса, и еще один тест для проверки средств обеспечения безопасности системы. В состав тестового набора могут быть включены, с одной стороны, тесты, которые работают с экранами ввода специфических данных или с экранами для определения запросов. Тестовый набор отображается на одно конкретное требование или на некоторый набор логически связанных требований, в то время как более детализированные тестовые случаи отображаются на элементы функциональной спецификации системы. Более подробно вопросы структурирования тестов рассматриваются в разделе "Архитектура тестов" в главе 3.

И последнее, что необходимо сказать о модульности, это то, что она способствует обнаружению дефектов на ранних стадиях цикла разработки. Теоретически каждый из уровней, на которые разбита система, должен проверяться на предмет присутствия дефектов, а только после этого становится возможным переход на следующий уровень. На практике план проведения испытаний может пересматриваться и корректироваться перед переходом на следующий уровень, коим является проектирование тестов. Проекты тестов могут пересматриваться и корректироваться до того, как начнется работа над детализированными тестовыми случаями. Эти промежуточные контрольные точки препятствуют распространению дефектов рабочих вариантов тестов далее по жизненному циклу тестирования, что может служить причиной возникновения проблем, приводящих к срывам рабочего графика.

## ОДИН КРУПНЫЙ ТЕСТ? — БОЛЕЕ ПОДРОБНО ОБ АРХИТЕКТУРЕ ТЕСТОВ

В принципе, можно разработать гигантский тест, который покрыл бы все аспекты тестируемого программного продукта, однако существует несколько причин, в силу которых эту идею нельзя реализовать на практике. Предположим, что в результате прогона этого гигантского теста обнаружено сразу несколько дефектов. Когда разработчик предпринимает попытку воспроизвести один из этих дефектов, дабы отыскать основную причину возникшей проблемы и устранить ее, то как же ему поступить в такой ситуации? Неужели снова выполнять прогон этого большого теста? Предположим, что всеми правилами и неправдами ему все-таки удалось устранить неисправность. Как теперь проветрить, что дефект и в самом деле устранен — опять-таки прогонять тот же самый супертест? Очевидно, что это явно не самый эффективный подход.

Поэтому лучше всего выбрать такую структуру, чтобы каждый тест охватывал конкретную часть функциональных возможностей тестируемого программного продукта. Если прогон такого теста завершается неудачей, то разработчику нетрудно будет еще раз прогнать этот тест и воспроизвести дефект, затем провести анализ результатов тестирования и устранить неисправность. В идеальном случае объем такого теста сводится до минимального набора действий, необходимого для выявления неисправности.

Если в основу тестирования программного продукта положены предъявляемые к нему технические требования, то эмпирическое правило для определения объема теста состоит в том, что на каждое техническое требование должен приходиться, по меньшей мере, один тест. Принцип "по меньшей мере, один тест на одно техническое требование" не только позволяет выбирать размерность теста в разумных пределах, но и поддер'живает разработку плана проведения испытаний. На базе пересмотра перечня технических требований можно оценить, сколько новых тестов потребуется для тестирования нового программного продукта. Как только вы начнете создавать тесты, руководствуясь этим принципом, вы сможете подсчитать, сколько времени потребуется на разработку и прогон теста на основании ранее полученных результатов.

Тестирование на базе технических требований позволяет качественно организовать испытания. Технические требования обычно объединяются в группы по функциональным свойствам. Свои тесты можно организовать по тому же принципу. Тесты, относящиеся к подготовке к работе технических средств заказчика, можно объединить в одну группу, равно как и тесты, предназначенные для проверки средств установки и наращивания возможностей программного продукта. Как следует из главы 3, группы логически связанных .тестов могут быть помещены в один *тестовый набор (test suite).*

Для тестирования конкретного технического требования может понадобиться более одного сценария. Например, если вы тестируете техническое требование, регламентирующее ввод данных заказчика, то требуемый тип данных и их количество может меняться в зависимости от того, принадлежит ли заказчик к "серебряному классу", "золотому классу" или "платиновому классу". Тест для проверки этого требования придется модифицировать в соответствии с классом заказчика.

**Один из способов разрешения подобного рода ситуаций заключается в создании трех различных тестовых случаев, по одному для каждого класса заказчика. *Тестовый случай (test case)* представляет собой совокупность входных данных теста, условий выполнения и ожидаемых результатов, которые разработаны для конкретной цели. Тестовый случай представляет наименьшую единицу тестирования, которую можно самостоятельно выполнить от начала до конца.**

***Врезка 4.1.***

## Определение целей теста

Самой первой задачей, с которой начинается проектирование теста, является строгая формулировка его целей или назначения. Предположим, что вы разрабатываете тесты для приложения TMT, описание которого приводится в главе 2 и в третьей части книги. Одно из технических требований этого приложения выглядит следующим образом (другие требования представлены на рис. 2.4 в главе 2):

> 2.2.1. Приложение должно предоставить средства создания, модификации, просмотра, хранения и поиска документов, имеющих отношение к плану проведения испытаний.

Техническое требование 2.2.1 распространяется на широкий набор функциональных свойств, но все они относятся к управлению документами, содержащими план проведения испытаний. Один из способов свести воедино тесты, относящиеся к этому требованию, предусматривает создание набора под названием "Test_Plans" ("Планы_проведения_испытаний") и сбор в нем всех тестов, предназначенных для проверки требования 2.2.1. Например, перед тестами, входящими в этот набор, можно поставить следующие цели:

1. Проверить, что программа предоставляет квалифицированному пользователю средства создания всех действующих типов документов, которые имеют отношение к плану проведения испытаний.
2. Проверить, что квалифицированный пользователь может сохранять и отыскивать любой составленный план проведения испытаний.
3. Проверить, что программа предоставляет квалифицированному пользователю средства модификации всех документов с планом проведения испытаний, которые были составлены и сохранены в памяти.
4. Проверить, что квалифицированный пользователь может отыскать и пересмотреть любой документ, содержащий план проведения испытаний, который был составлен и сохранен в памяти.

Каждая цель теста устанавливает, *что* тест будет делать, но отнюдь не то, *как* он это будет делать. *Как* — это дело подробной методики тестирования, которая разрабатывается в виде части соответствующего случая тестирования. Цели теста представляют собой уточнение плана проведения испытаний, другими словами, тест должен быть ориентирован на проверку технического требования 2.2.1. Другим отличием проекта теста от тестового случая является то, что проекты тестов обычно создаются с использованием спецификации технических требований. Тестовые случаи, требующие более подробного знакомства с программным продуктом, требуют, по

**меньшей** мере, функциональной спецификации или, возможно, дополнительной информации из проектной документации по программному продукту.

Обратите внимание на то обстоятельство, что примеры целей тестирования требует некоторой разъяснительной информации, когда дается определение следующего "слоя луковицы". Цель номер 1 теста, например, не дает определения "квалифицированного пользователя" — т.е. уточнения, которое зависит от того, как спроектировано управление доступом к программе. Должны быть построены тесты, проверяющие, могут ли получить доступ к данным квалифицированные пользователи, и в то же время неквалифицированные пользователи не могут. Цели теста также ничего не говорят о том, что собой представляют "действующие типы документов с планом проведения испытаний", однако наличие этой формулировки подсказывает специалисту по тестированию, что нужно рассмотреть в качестве случаев как действующих, так и не действующих типов документов, если установлены конкретные критерии применимости.

По сути дела, в процессе формулирования целей тестирования следует пользоваться следующими рекомендациями:

- Четко сформулировать назначение каждого теста.
- Определить модульную структуру для тестов, так чтобы каждый тест имел единственную цель, и чтобы его можно было отобразить на помеченное требование в спецификации технических требований. Однако имейте в виду, что для тестирования некоторого конкретного требования может понадобиться более одного теста.
- Указать, что должен делать тест, но в то же время не следует объяснять, как это достигается — это должно быть указано в тестовом случае.

## Определение спецификаций ввода

Спецификации ввода регламентируют форматы файлов входных данных, записей баз данных, конфигурационных файлов, оборудования или других входных данных, необходимых для того, чтобы привести испытательную систему в требуемое состояние для выполнения тестов. Спецификации ввода не охватывают ввод с клавиатуры или ввод при помощи мыши, который производится в процессе тестирования; такие виды ввода должны описываться в подробных методиках испытаний.

Каждому входному файлу или образу должен быть присвоен уникальный идентификатор, причем они должны храниться в среде, обеспечивающей управление версиями и регулярное дублирование. Хранилище входных данных должно быть описано в плане проведения испытаний, а в спецификации ввода теста должны присутствовать ссылки на это хранилище. Например, набор документов, содержащих план проведения испытаний, может понадобиться для тестов, описанных в предыдущем разделе. Они могут иметь имена TP_inputl.doc, TP_input2.doc и т.п., и храниться в каталоге, например, D:\Test\Project_Name\Test_Plan\Inputs.

## Определение конфигурации средств тестирования

Прогон каждого теста должен выполняться в известной среде тестирования, чтобы результаты прогона были предсказуемыми и воспроизводимыми. Это означает, что конфигурация аппаратных средств, операционная система, версия тестируемого

программного продукта и начальное состояние системы должны быть определены. Как отмечалось в главе 3, работа по составлению плана проведения испытаний должна предусматривать анализ различных вариантов конфигурации средств тестирования и выбор из них конфигураций, пригодных для системных испытаний.

Задача заключается в том, чтобы на стадии проектирования конкретного теста выбрать одну или несколько конфигураций, которые можно использовать для достижения целей этого теста. Зачастую один и тот же тест нужно выполнить на некотором множестве конфигураций, чтобы смоделировать различные рабочие условия заказчика. Например, крупный заказчик может использовать преимущественно стандартные конфигурации операционной системы, процессора, накопителя на жестких дисках, сетевой операционной системы и оперативной памяти — такую конфигурацию можно смоделировать в лабораторных условиях с высокой степенью точности.

Один из способов отслеживания конфигураций предусматривает их описание в плане проведения испытаний с присвоением каждой конфигурации уникальных идентификаторов. В таком случае в плане проведения испытаний гораздо проще ссылаться на одну или несколько избранных конфигураций.

## Документ проектов тестов

Назначение документа проектов тестов состоит в том, чтобы собрать в одном месте всю информацию, порожденную в результате различных видов тестовой деятельности. Документ проектов тестов может быть представлен в виде электронной таблицы, в виде таблицы текстового процессора или в виде базы данных. Если вы пользуетесь инструментальным средством автоматизированного управления требованиями, его можно применить для сбора информации о проекте теста. Пример записи в документе проектов тестов показан в таблице 4.1.

**Таблица 4.1. Пример записи в проектном документе тестов**

| Идентификатор определения требования | Идентификатор системного случая тестирования | Входные данные теста | Конфигурация теста | Назначение теста |
|---|---|---|---|---|
| RD2.2.1 | ST2.2.4 | TP_Input1.doc | TC2.0 | T02.2.4. Проверить, что квалифицированный пользователь может отыскать и ознакомиться с любым ранее созданным документом, содержащим план проведения испытаний. |

Эта таблица в какой-то степени похожа на матрицу RTM (Requirement Traceability matrix — матрица прослеживаемости требований), которая рассматривалась в главе 2 (см. рис. 2.5). Два первых столбца таблицы должны соответствовать вхождениям в матрицу RTM, остальные столбцы соответствуют данным, описанным в трех предыдущих разделах данной главы.

Как только проектный документ тестов будет готов, его нужно проверить в контексте связанных с ним материалов: документа определения требований (техниче-

ского задания), определения входных данных теста и конфигурации средств тестирования. Цели такой проверки связаны с необходимостью убедиться:

- Что рассматриваемым документом проектов тестов были охвачены все технические требования. Если то или иное требование не тестируется, в матрице RTM или в документе проектов тестов должна присутствовать запись, поясняющая, почему соответствующее требование не покрывается тестом.
- Что каждый тестовый случай поддерживается соответствующими входными данными.
- Что для каждого тестового случая имеется подходящая конфигурация, причем эти конфигурации не являются избыточными.

Документ проектов тестов служит основой для разработки детальной методики тестирования. В результате его пересмотра тесты могут распределяться среди исполнителей группы тестирования для дальнейшей детализации.

# Разработка тестовых случаев

Тестовый случай представляет собой основной компонент динамического тестирования. По сути дела, этап системного тестирования — едва ли что-то большее, нежели просто прогон тестовых случаев на некоторой последовательности программных сборок с целью обнаружения и устранения дефектов. Это означает, что основная обязанность специалиста по тестированию заключается в написании и прогоне тестовых случаев. В этом разделе мы обсудим, как проектировать и создавать высококачественные тестовые случаи.

В главе 1 было дано определение тестирования программного обеспечения как процесса анализа или эксплуатации программного обеспечения с целью выявления дефектов. *Тест (test)* представляет собой набор операций, предназначенных для получения одного или большего числа ожидаемых результатов в некоторой программной системе. Если получены все ожидаемые результаты, считается, что тест прошел (т.е. выполнен успешно). Если фактический результат отличается от ожидаемого, считается, что тест не прошел (т.е. завершился неудачно).

Первое, что следует отметить в приведенном определении, так это то, что каждый тест состоит из двух компонентов: (1) совокупность выполняемых вами действий и (2) последовательность событий, которые должны произойти в результате этих действий. Выполняемые действия суть тестовые действия, которые в совокупности образуют методику тестирования. Последовательность событий, происходящих в результате этих действий, называются ожидаемыми результатами. Чтобы тест был эффективным, должны быть четко и однозначно определены как методика, так и ожидаемые результаты.

Во-вторых, если методика тестирования и ожидаемые результаты определены правильно, тест должен давать результат, по которому можно сделать однозначный вывод относительно успеха или неудачи испытания. При вводе в программу двух чисел с целью получения их суммы тест считается пройденным, если на выходе программы будет получен корректный результат; в противном случае тест рассматривается как не пройденный.

Как отмечалось ранее, для удобства выполнения тесты можно и далее разбивать на тестовые случаи. Если некоторый тест требует выполнения пространной методики тестирования с множеством ожидаемых результатов, имеет смысл разбить такой тест на тестовые случаи. Однако при этом следует иметь в виду, что тестовый случай есть наименьший модуль тестирования, и что с каждым тестовым случаем должен быть связан, по меньшей мере, один ожидаемый результат.

В силу того, что целью тестирования является выявление дефектов, хорошим тестом считается тест с высокой вероятностью обнаружения дефекта. Чтобы спроектировать тест, обладающий высокой вероятностью обнаружения дефекта, специалист по тестированию должен стать на позицию "конструктивного разрушения" программного продукта. "Конструктивное разрушение" означает выявление проблем в программном продукте с целью их устранения. Сложность при этом связана с тем, что в задачи специалиста по тестированию не входит выявление обстоятельств, при которых программный продукт работает; наоборот, тестировщик пытается обнаружить такие ситуации, при которых продукт *не* работает. При проектировании тестового случая важно не делать никаких предположений относительно того, что та или иная функция программного продукта работает исправно. Вы не должны рассчитывать на то, что какая-либо сборка программного кода будет установлена правильно, или на то, что структура базы данных соответствует проекту, или что программа самостоятельно восстановится после потери напряжения в электросети. Нельзя также рассчитывать и на то, что если программа успешно работает на заданной платформе, то она будет работать столь же хорошо и на компьютере с меньшим пространством памяти или под управлением другой операционной системы.

Другое свойство хорошо спроектированного теста — его повторяемость. Если прогон теста завершился неудачей, очень важно воспроизвести точные условия, при которых это произошло. По этой причине важно, чтобы начальное состояние системы было определено тестовым случаем в терминах используемой версии программного продукта, конфигурации аппаратных средств, количества пользователей системы и т.д. Важно также, чтобы была известной точная методика, используемая при выявлении неисправностей. В идеальном случае должна фиксироваться точная последовательность нажатых клавиш, щелчков мыши или другие события, которые привели к отмеченным отклонениям от ожидаемых результатов. На практике все эти подробности могут оставаться неизвестными, возможно, из-за того, что методика тестирования не расписана во всех подробностях, или из-за того, что произошло какое-то неконтролируемое случайное событие без ведома тестировщиков. Вот почему, когда происходит сбой, важно, не жалея времени, немедленно воспроизвести неисправность, подробно фиксируя при этом действия, которые привели к сбою, если только они явно не расписаны в методике испытаний.

Хорошо спроектированный тест снабжен одним или большим числом четко определенных ожидаемых результатов и четко определенными критериями успешных/неудачных испытаний. Обычно ожидаемые результаты и критерии успешных/неудачных испытаний тесно связаны. Если ожидаемый результат или некоторый набор ожидаемых результатов будет получен, когда система переводится из одного заданного состояния в другое, то тестовый случай проходит. Если ожидаемые результаты не удается зафиксировать после выполнения заданных действий, тест не проходит. Например, предположим, что в поле ввода данных вводится значение поч-

тового индекса, и при этом ожидается, что на экран будет выведено название соответствующего штата после щелчка на кнопке "Укажите штат". Если после ввода некоторого набора почтовых индексов для каждого из них появляется правильное название штата, это значит, что тест получает ожидаемые результаты и поэтому проходит. Если же при вводе почтового кода Сиэтла на экране отображается штат Флорида, считайте, что вам повезло в плане обнаружения дефекта, поскольку ожидаемый результат не был получен.

Следует отметить, по меньшей мере, один дополнительный признак хорошо спроектированного тестового случая — он не должен быть избыточным. Оптимальный рабочий график тестирования требует, чтобы вы выполнили объем тестирования, достаточный для обнаружения всех неисправностей перед поставкой программного продукта заказчику, но вы не должны тратить время на прогон избыточных тестов. В рассмотренном выше примере вы не должны тратить время на отображение на экране названия штата для каждого известного почтового индекса, если эта функция не критична для бизнес-операций заказчика. Если с отображением на экране почтовых индексов связана крупная неприятность, наверняка потребуется потратить время на тестирование функции отображения. Кроме того, следует задуматься над вопросами автоматизации этой утомительной задачи. Но если это свойство реализуется из соображений удобства, возможно, потребуется лишь проверить какой-нибудь допустимый и несколько недопустимых индексов, дабы подтвердить правильность базовых функциональных свойств. Эта тема обсуждается в разделе "Разбиение на эквивалентные классы" далее в этой главе, а также в главе 10.

Резюмируя, можно утверждать, что с тестовым случаем должны быть связаны следующие признаки:

- Он должен обладать высокой вероятностью обнаружения дефекта.
- Он должен быть воспроизводимым.
- Он должен обладать четко определенными ожидаемыми результатами и критериями успешного или неудачного выполнения теста.
- Он не должен быть избыточным.

## Разработка детализированных методик тестирования

Детализированные методики тестирования могут быть разработаны на основе проектов тестов. Уровень детализации методик тестирования, представленных в письменном виде, зависит от квалификации и знаний исполнителей, которые выполняют прогон тестов. Вы планируете привлечь к выполнению системного испытания программного продукта исполнителей, мало знакомых с этим продуктом? Если это так, то они должны пройти определенную подготовку, а методика тестирования должна быть четко сформулирована с тем, чтобы они выполняли корректные тестовые действия. В случае, когда тестировщики обладают подробными знаниями программного продукта, методика тестирования может быть менее детализированной. Решение относительно того, каким должен быть уровень детализации методик тестирования, имеет большое значение, ибо можно напрасно потратить время на описание излишних подробностей для подготовленного пользователя. С другой стороны, напрасно потратить время можно и на обучение неподготовленных тестировщиков тому, как

проводить испытания, если методика не содержит необходимых деталей. Так или иначе, желательно найти разумный компромисс.

Отдельного рассмотрения заслуживает один специальный случай. Если тест заслуживает того, чтобы быть автоматизированным, целесообразно выделить время на предварительную разработку детализированной методики тестирования, с помощью которой инженер по автоматизации сможет однозначно сформулировать задачу автоматизации. Расплывчатая методика тестирования, скорее всего, приведет к появлению неточных или неэффективных автоматизированных тестов. Рекомендуемый уровень детализации методики тестирования прояснится после дальнейшего обсуждения ожидаемых результатов.

Существует широкий выбор технологий, которые могут использоваться для тестирования программных продуктов. В этой главе мы будем рассматривать технологии тестирования методом черного ящика. Тестирование методом черного ящика представляет собой тестирование на системном уровне, которое имеет дело только со "внешними" аспектами программы. Тестирование методом черного ящика не предполагает каких-либо знаний о внутреннем функционировании программного продукта и проводится с использованием только внешних интерфейсов, таких как пользовательские интерфейсы или интерфейсы API (Application Programming Interface — интерфейс программирования приложений). Более подробный анализ тестирования методом черного ящика можно найти в [17], [15] и [27]. Более подробную информацию о тестировании с использованием метода черного ящика можно найти в главе 10 настоящей книги.

Двумя широко распространенными технологиями разработки тестов для тестирования методом черного ящика являются разбиение на классы эквивалентности и анализ граничных значений. Обе технологии помогают уменьшить общее количество тестовых случаев или проверяемых условий, которые необходимы для полного покрытия функциональных возможностей программы. Вполне понятно, что эти методы являются ценной частью понятия быстрого тестирования, ибо чем меньше требуется разрабатывать и прогонять тестов с целью получения того же функционального покрытия, тем эффективнее становится тестирование.

**Разбиение на классы эквивалентности.** Разбиение на классы эквивалентности представляет собой технологию проектирования тестов, ориентированную на снижение общего числа тестов, необходимых для подтверждения корректности функциональных возможностей программы. Основная идея, стоящая за разбиением на классы эквивалентности, заключается в том, чтобы разбить область ввода программы на классы данных. Если проектировать тесты для каждого класса данных, но не для каждого члена класса, то общее количество требуемых тестов уменьшается.

В качестве примера рассмотрим программу отображения почтовых индексов, которая рассматривалась ранее в главе. Предположим, что когда пользователь вводит пятизначный почтовый индекс и общую массу отправляемого груза в унциях, программа возвращает стоимость доставки пакета. Областью ввода для этой программы являются почтовые индексы и масса брутто отправляемого груза. Область ввода почтового кода может быть разбита на класс допустимых вводов и класс недопустимых вводов следующим образом:

- Допустимыми вводами являются все пятизначные наборы цифровых символов, образующих рабочий почтовый код
- Недопустимыми вводами являются:
  - Наборы цифровых символов, содержащие менее пяти символов
  - Наборы цифровых символов, содержащие более пяти символов
  - Наборы из пяти символов, не являющиеся рабочим почтовым кодом
  - Наборы из нецифровых символов.

Аналогичное разбиение может быть построено и для массы брутто отправляемого груза. Например, требуется программа, работающая только с грузами, масса которых находится в диапазоне от 1 до 100 унций. В этом случае числовые значения, попадающие в диапазон от 1 до 100, включая и конечные точки, являются допустимыми. Все значения меньше 1 и больше 100, отрицательные значения и нецифровые значения образуют недопустимые классы ввода.

Нужно спроектировать такие тесты, которые выполняют проверку, по меньшей мере, одного представителя каждого допустимого класса ввода и, по меньшей мере, одного представителя недопустимого класса ввода. В результате тестирования с применением допустимых вводов должны быть получены однозначно определенные и ожидаемые результаты. Для почтового кода 78723 и массы в 20 унций ожидается получить конкретное значение стоимости, и это значение должно быть определено в функциональных требованиях. В случае ввода недопустимых данных должно быть получено соответствующее сообщение об ошибке, если оно определено в технических требованиях и спецификациях. При вводе недопустимых данных программа, по меньшей мере, не должна завершаться аварийно, вызывать искажение данных или вести себя непредсказуемым образом.

Более подробное обсуждение разбиения на классы эквивалентности можно найти в главе 10, а также в [36], [27], [43], [28] и [17].

**Анализ граничных значений.** Анализ граничных значений представляет собой технологию проектирования тестов, которая является дополнением разбиения на классы -эквивалентности. Вместо того чтобы выбирать некоторый конкретный элемент класса эквивалентности, анализ граничных значений предлагает проектировщику теста выбрать элементы, которые находятся "на границе" класса. Экспериментально было доказано, что дефекты имеют тенденцию концентрироваться на границе области ввода, а не в ее центре. Не особенно ясно, почему так получается, это всего лишь установленный факт.

Например, в случае программы, которая для почтового индекса и веса отправляемого груза вычисляет стоимость доставки, анализ граничных значений позволяет применять в качестве тестовых значений минимальное и максимальное значение веса (1 унция и 100 унций), а также ближайшее значение, меньшее минимально допустимого (0 унций), и ближайшее значение, большее максимально допустимого (101 унция). Эти значения позволяют проверить границы диапазона допустимых значений, а также значения, выходящие за пределы этого диапазона. Более подробную информацию по анализу граничных значений можно найти в главе 10, а также в [36], [27], [43] и [17].

## Определение ожидаемых результатов

Основу динамического тестирования составляет прогон управляемого набора операций на конкретной сборке программного продукта и сравнение полученных результатов с ожидаемыми результатами. Если получены ожидаемые результаты, то тест считается прошедшим на этой сборке; если зафиксировано аномальное поведение, **то** считается, что тест на этой сборке не прошел, в то же время этот тест, возможно, позволил обнаружить очередную неисправность. Непременным условием для определения, прошел или не прошел конкретный тест, является однозначное определение ожидаемых результатов этого теста.

Таблица 4.2. Примеры ожидаемых результатов

| **Шаг** | **Действие** | **Ожидаемый результат** | **Отметка** (V) |
|---|---|---|---|
| 1. | Щелкнуть на элементе "Стоимость доставки" в главном меню. | На экран выводится меню Стоимость доставки. | |
| 2. | Ввести значение "101" в поле веса доставляемого груза. | Сообщение об ошибке "Неправильно указан вес доставляемого груза". | |
| 3. | Ввести значение "0" в поле веса доставляемого груза. | Сообщение об ошибке "Неправильно указан вес доставляемого груза". | |
| 4. | Ввести значение "100" в поле веса доставляемого груза. | На экран выводится "100 унций" как вес доставляемого груза | |

Фрагмент тестового случая для программы, которую рассматривается в качестве примера и вычисляет стоимость доставки по почтовому индексу груз с заданным весом, приводится в таблице 4.2. Шаги методики тестирования пронумерованы в первом столбце. Краткие описания действий, выполняемых тестировщиком, даны во втором столбце, а исчерпывающие описания ожидаемых результатов находятся в третьем столбце. Предполагается, что в этом примере при проведении испытаний будет использоваться твердая (или интерактивная) копия этой таблицы, в связи с чем в ней предусмотрен четвертый столбец, в котором тестировщик может отмечать каждый выполненный шаг. В дальнейшем мы покажем, что следует предпринимать в случае, если какой-то тест не проходит. Особое внимание следует обратить на то, чтобы на каждом шаге методики тестирования предоставлялись четкие описания ожидаемых результатов.

## Установка и очистка — тестирование из известного состояния

Один из основных принципов тестирования заключается в том, что всегда необходимо знать, в каком состоянии находится тестируемая система. Если обнаружен дефект, но тестировщик не знает, какие действия привели к сбою, в таком случае возникают трудности при воспроизведении этого сбоя. Необходимость проводить тестирование из известного состояния означает, что каждый тестовый случай должен привести аппаратные и программные средства в известное состояние. Это отнюдь не означает, что тестировщик обязан обесточить систему, перезагрузить ее и запускать программу с самого начала — как и в любых других ситуациях при тестировании, все имеет свои пределы.

Во избежание перехода испытываемой системы в состояния, которые невозможно воспроизвести, существует ряд практических мер. Одной из них является выполнение энергетического цикла системы перед началом тестирования — это, по-видимому, следует делать в начале каждого рабочего дня. Другая мера предусматривает периодическое обновление баз данных и удаление тестовых каталогов. Если применяется автоматизация тестирования, то очистку баз данных и каталогов данных можно проводить часто и не тратить на это много времени. Возможно, что тестируемая система производит изменения во встроенных программных средствах. Если это так, то встроенные программы потребуется перезагрузить либо до запуска логически связанных тестов, либо в начале каждого рабочего дня, что обеспечит тестирование неискаженной версии встроенных программных средств. В идеальном случае для сохранения текущего состояния тестируемой системы в специальном хранилище, для регистрации различного рода отклонений и для перевода тестировщиком системы в нужное состояние используются автоматизированные инструментальные средства.

Существуют два подхода к инициализации теста. Один из них предусматривает применение специальной программы настройки, которая перед началом тестирования приводит систему в известное состояние, и использование стандартных программ очистки, которые "аннулируют" изменения, внесенные в процессе тестирования. Другой подход предусматривает только настройку и никакой очистки. Если перед прогоном каждого теста выполняется соответствующая операция настройки, то не имеет значения, что произойдет в конце теста — все равно до прогона следующего теста известные условия будут восстанавливаться, при этом усилия, затраченные на очистку, могут оказаться напрасными. Разумеется, могут возникнуть проблемы в ситуациях, когда добавляется новый тест, не рассчитанный на настройку, а выполненные ранее тесты перевели систему в непредусмотренное состояние.

Рекомендуемый в таких случаях практический метод предусматривает выполнение очисток после прогона теста, который внутренне переводит систему в непредусмотренное состояние. Например, если вы переполняете каталог данных, искажаете базу данных или переводите систему в состояние сбоя, то наилучшим решением будет восстановление системы в ее нормальном состоянии по окончании теста. Аналогично, .если осуществляется прогон теста, о котором известно, что он зависит от начального состояния системы, то лучше всего будет воспользоваться установочной процедурой для инициализации тестируемой системы.

## Шаблон тестового случая

Все предложения, касающиеся проектирования тестового случая, обсуждавшиеся до сих пор, могут быть сведены в единый шаблон тестового случая. Возможно, возникнет желание создать собственный шаблон, тем не менее, стоит обратить внимание на один из вариантов такого шаблона, показанный на рис. 4.2. Тестовые случаи, основанные на этом шаблоне, можно распечатать для целей неавтоматизированного тестирования, либо за счет организации отображения в браузере их HTML-версий появится возможность прогона тестов в интерактивном режиме.

| **Информация о тестовом случае** | | | |
|---|---|---|---|
| **Идентификатор тестового случая** | SC03 ver3.0 | | |
| **Владелец теста** | Джин Дуглас (Jean Douglas) | | |
| **Местонахождение теста (путь)** | TestServer:D:\TestProject\TestSuite\SC03.doc | | |
| **Дата последнего пересмотра** | Месяц/день/год | | |
| **Тестируемое требование** | SC101 | | |
| **Конфигурация средств тестирования** | ST02 | | |
| **Взаимозависимость тестовых случаев** | Выполнить прогон теста SC01 и его настройку перед прогоном данного теста. | | |
| **Цель теста** | Проверить, что допустимые входные значения веса отправляемого груза дают правильные значения стоимости доставки, и что недопустимые входные значения приводят к выдаче сообщений об ошибке. | | |
| **Методика тестирования** | | | |
| **Настройка на прогон теста** | Не проводится | | N/A |
| | | | |
| **Шаг** | **Действие** | **Ожидаемый результат** | **Отметка (V)** |
| 1. | Щелкнуть на элементе "Стоимость доставки" в главном меню. | Отображается меню "Стоимость доставки" | **(V)** |
| 2. | Ввести "101" в поле веса доставляемого груза | Сообщение об ошибке "Неправильно указан вес доставляемого груза" | **(V)** |
| 3. | Ввести "0" в поле веса доставляемого груза | Сообщение об ошибке "Неправильно указан вес доставляемого груза" | **X** |
| 4. | Ввести "100" в поле веса доставляемого груза | Указан вес доставляемого груза "100 унций" | **(V)** |
| 5. | Ввести "1" в поле веса доставляемого груза | Указан вес доставляемого груза "1 унция" | **(V)** |
| | | | |
| **Очистка после прогона теста** | Не производится | | **N/A** |
| **Результаты теста** | | | |
| **Тестировщик: JD** | **Дата прогона теста:** месяц/день/год | | **Результат теста (P/F/B): F** |
| **Примечания:**<br>- Тест потерпел неудачу на шаге 3.<br>- При возникновении неисправности выдается код ошибки BR1011. | | | |

*Рис. 4.2. Пример тестового случая*

Основными элементами шаблона тестового случая являются:

- Идентификатор тестового случая — включает номер версии теста.
- Владелец теста — имя или инициалы лица, эксплуатирующего тест (оно может не совпадать с именем автора теста).
- Дата последнего пересмотра — эта информация позволит определить, является ли тест актуальным.
- Наименование теста — описательное имя теста, которое позволяет легко отыскать тест и понять его назначение. Применение имен, не несущих смысловой нагрузки, например, "xxxLLL0123.tst", не рекомендуется.
- Местонахождение теста — это полное имя пути, включая сервер.
- Тестируемое техническое требование — это должен быть уникальный идентификатор, который отображается на документы с техническими требованиями.
- Цель тестирования — краткая и четкая формулировка того, чего должен достичь данный тест. Более подробная информация изложена выше, в разделе "Определение целей теста".
- Конфигурация средств тестирования — спецификация ввода, спецификация вывода, условия испытаний.
- Настройка на прогон теста— эта процедура подобна методике тестирования. Она предусматривает описание действий, выполняемых тестировщиком, и ожидаемых результатов. Если настройки автоматизированы, это может выглядеть как **run setupSC03.pl.**
- Методика тестирования — описание действий, выполняемых тестировщиком, и ожидаемых результатов.
- Взаимозависимость тестовых случаев — идентификация любого тестового случая, прогон которого должен предшествовать прогону данного теста, дабы выполнение данного теста начиналось при заданных условиях.
- Очистка теста — если системы была переведена в неустойчивое состояние или данные оказались разрушенными, очистка предоставит шанс устранить подобные ситуации.

## Управление конфигурацией тестового случая

Во время прогона теста необходимо выполнить заданный набор операций на заданной конфигурации системы. Это обстоятельство гарантирует, что тест можно воспроизвести и что он осуществляет проверку заданных функциональных возможностей. По мере того, как меняется программный продукт в результате исправления обнаруженных дефектов или по причине изменений, внесенных в технические требования, тесты также требуют изменений. Функциональные возможности программного продукта и тесты должны синхронно реагировать на такие изменения, в противном случае тесты приведут к неверным результатам.

Управление направлением изменения программного продукта осуществляется благодаря использованию инструментальных средств CM (Configuration Management— управление конфигурациями), таких как инструментальные средства CVS (Control Version System — система управления версиями) или ClearCase. С помощью

упомянутых средств файлы, содержащие программные коды, проверяются при сохранении и при модификации. Эти средства обеспечивают управление версиями, при этом предыдущие версии файла архивируются и, в случае необходимости, легко могут быть найдены.

Такой тип СМ-систем очень хорошо работает как в условиях неавтоматизированных, так и автоматизированных тестовых случаев. Он позволяет идентифицировать некоторый набор тестов так, что с некоторой группой тестов ассоциируется номер версии. Например, версия 3.0 набора тестов по вводу данных может использоваться для испытаний версии 3.0 тестируемого программного продукта.

# Пересмотр и отладка тестов

После написания или автоматизации тест необходимо проверить на наличие дефектов с целью их немедленного устранения, и затем испытать на некоторой сборке программного продукта. Статическое тестирование тестовых случаев можно проводить **с** применением той же технологии, которая используется для проверки выполнения технических требований, т.е. обследования, сквозного контроля или экспертных оценок (см. раздел "Методы статического тестирования" в главе 2). Эти методы могут применяться в различных сочетаниях, если сложность тестов такова, что необходимо тщательное статическое тестирование. Например, экспертные оценки можно использовать в методиках, а формальное обследование — в автоматизированных сценариях, реализующих методики тестирования.

При проверке тестовых случаев необходимо обратить внимание на ряд следующих моментов:

- В какой мере соответствует тест или тестовый набор функциональным возможностям, заявленным в технических требованиях?
- Покрывает ли тестовый набор все технические требования?
- Организованы ли тесты достаточно эффективно, чтобы можно было обходиться минимальными конфигурациями средств тестирования?
- Включены ли тесты в систему управления конфигурациями?
- Есть ли среди тестов избыточные? Можно ли устранить эту избыточность?
- Достаточно ли подробно разработана методика тестирования, чтобы стала возможной ее автоматизация?
- Снабжен ли каждый шаг тестирования четко определенными ожидаемыми результатами (критерий удачного/неудачного исхода испытаний)?
- Правильно ли воспроизводят автоматизированные тесты неавтоматизированные шаги тестирования?

Если вы провели статическое тестирование плана проведения испытаний, то пересмотр тестовых случаев пойдет намного проще, поскольку вопросы, касающиеся покрытий технических требований тестами, должны решаться во время пересмотра плана проведения испытаний. Вопросы, касающиеся избыточности тестов и эффективного использования конфигурации средств тестирования, также должны быть затронуты в процессе пересмотра плана проведения испытаний, во всяком случае, на уровне предварительных решений. Тщательные пересмотры планов проведения ис-

пытаний и тестовых случаев, равно как и статическое тестирование программного продукта, содействуют раннему обнаружению дефектов в тестах и позволяют сократить затраты времени на отладку тестовых случаев.

В рамках такого пересмотра каждый новый тест нужно прогнать на некоторой сборке программного продукта, дабы удостовериться, что методика тестирования позволяет получить ожидаемые результаты. Поскольку этот тест, по-видимому, будет прогоняться на программном коде, в изобилии содержащем дефекты, необходимо соблюдать определенную осторожность при анализе неудачных исходов прогона теста в плане, что является источником проблемы — программный код или сам тест. Отладка тестового случая часто требует от исполнителей такого же инженерного искусства и аналитических способностей, которые так необходимы разработчикам программного обеспечения во время отладки кода программного продукта.

## Автоматизация тестовых случаев

В главе 3 отмечалось, что при правильном планировании и разумных ожиданиях использование автоматизированных инструментальных средств и автоматизированных тестовых случаев представляет собой хороший способ снижения затрат времени на тестирование программного продукта. В этой главе мы кратко обсудим некоторые руководящие принципы, регламентирующие автоматизацию тестовых случаев. Автоматизация тестов — это вид деятельности по разработке программного обеспечения, для выполнения которого нужно обладать таким же опытом и квалификацией, как и при создании любого другого программного продукта. Если вы планируете потратить средства и усилия на автоматизацию тестовых случаев, рекомендуем ознакомиться с материалами [15] и [17].

Ниже перечисляются некоторые руководящие принципы автоматизации тестовых случаев. Разумеется, приводимые ниже принципы следует привязать к конкретным условиям.

- Выполняйте автоматизацию только тех тестовых случаев, которые будут повторяться достаточно большое количество раз, чтобы такая автоматизация была рентабельной с точки зрения стоимости. Если вы намерены автоматизировать тесты, которые должны выполняться всего несколько раз, стоимость автоматизации в смысле времени и финансов может превзойти ожидаемую выгоду.
- Разрабатывайте сначала неавтоматизированную версию тестового случая и только после этого переходите к его автоматизации. Контрольные примеры должны быть отлажены с тем, чтобы убедиться, что неавтоматизированная методика испытаний определена верно, а конфигурация тестового случая работает правильно. Попытки одновременной отладки методики тестирования и программного кода, который автоматизирует эту методику, могут оказаться неэффективными, более того, в худшем случае они могут стать причиной появления ошибок в тестовом случае. Если неавтоматизированный тестовый пример подвергается отладке первым, он становится базисом, с которым можно сверять автоматизированный тестовый случай.
- Предназначенная для автоматизации неавтоматизированная методика тестирования должна быть четко сформулирована и должна допускать однозначное толкование с тем, чтобы инженер по автоматизации мог сосредоточиться на

разработке программного кода, а не задумываться над тем, как должен выполняться тот или иной тестовый шаг.

- Определите стандартную практику кодирования и придерживайтесь ее требований. Автоматизация представляет собой программное обеспечение, которое должно разрабатываться столь же аккуратно, как и тестируемый программный продукт. Если планируется использование автоматизированных сценариев в качестве регрессивных тестов, то вполне возможно, что они будут нужны в течение продолжительного времени. В конечном итоге может случиться так, что исполнитель, эксплуатирующий тестовый сценарий, и его разработчик — разные лица. В подобных ситуациях четко написанный и правильно задокументированный программный код есть непременным условием его эффективной эксплуатации.
- Автоматизированные тесты должны проверяться на соответствие неавтоматизированным тестам, от которых они произрастают. Пересмотры программных кодов сценариев приносят большую пользу при выявлении ошибок в автоматизации и при контроле соблюдения требований стандартов. Полная проверка автоматизированного сценария должна подтвердить, что он предоставляет те же функциональные свойства, что и его неавтоматизированный прототип, и эта проверка должна проводиться в лабораторных условиях. При этом выполняется параллельный прогон неавтоматизированной и автоматизированной версий теста на всех запланированных конфигурациях тестовых средств и в запланированных условиях.
- Автоматизированные тестовые сценарии должны находиться под управлением системы поддержки множества конфигураций в той же структуре каталогов, что и неавтоматизированные тесты. Система управления версиями или конфигурациями должна следить за тем, чтобы в каждый конкретный момент времени была активной только одна санкционированная версия сценария. Управление версиями обеспечивает воспроизводимость и повторяемость каждого теста. При воспроизведении дефектов, при проверке исправлений дефектов, а также при выполнении регрессивного тестирования очень важным условием является наличие единственного известного теста, ориентированного на выявление заданного дефекта.

# Что дальше

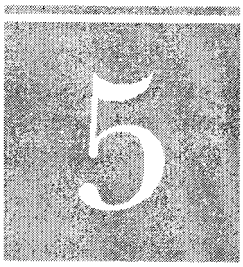
В этой главе обсуждались проектирование и разработка тестов. Ключевыми пунктами в процессе проектирования и разработки тестов являются:

- Определение целей теста (совершенствование подхода к тестированию и уточнение объема тестирования).
- Определение спецификаций ввода для каждого теста.
- Определение тестовой конфигурации для каждого теста.
- Проверка проектных решений тестов на предмет обеспечения необходимого уровня покрытия и технической точности.

- Разработка методики тестирования, обладающей требуемым уровнем детализации.
- Автоматизация часто используемых тестов, требующих больших затрат времени.
- Перевод тестовых примеров под управление конфигурациями.
- Определение настройки и очистки тестов с учетом их взаимозависимости.
- Пересмотр методик тестирования.
- Проверка автоматизированных тестов с использованием статических и динамических средств.

В результате выполнения всех этих действий будет получен набор отлаженных тестовых случаев, который может использоваться для проведения системных испытаний. В следующей главе мы рассмотрим, что требуется для прогона тестов, документирования выявленных дефектов и составления отчетов по результатам выполнения тестов.

# Системные испытания




**Темы, рассматриваемые в главе:**

- Обнаружение и отслеживание дефектов
- Прогон тестов
- Составление отчетов по результатам тестирования
- Критерий выхода из испытаний и готовность выпуска программного продукта
- Что дальше

Для профессионального тестировщиков программного обеспечения стадия системных испытаний — это то же, что игровой день для футболиста или день спектакля для театрального актера. Проделана большая работа по планированию, все подготовительные работы завершены, наступило время действовать. Как показано на рис. 5.1, первое, что нужно сделать, это проверить, все ли на месте и все ли готово. Все эти мероприятия могут быть дополнены применением критериев входа в системные испытания, определения которых включаются в план проведения испытаний. Критерий входа в испытания принимает форму опросного листа: составлен ли, пересмотрен и утвержден план проведения испытаний? Завершила ли группа разработчиков запланированное тестирование модулей и проверку взаимодействия и функционирования компонентов программного продукта? Проведены ли испытания "на герметичность" с целью удостовериться в том, что тестируемая программа может быть установлена и, по меньшей мере, способна продемонстрировать открытый экран?

Если достигнуто соответствие критериям вхождения в испытания, тестирование системы можно начинать. Начало тестирования обычно представляет собой важный этап графика проектных работ. В идеальном случае задержка системных испытаний преобразуется в задержку поставки программного продукта, хотя группу тестирования часто просят отыскать возможности ускорить испытания, чтобы программный продукт был поставлен во время. Несмотря на риск использования слишком многих аналогий, добавим еще одну: если сравнить разработку программного продукта с эстафетой, то группа тестирования пробегает последний этап гонки. Если бегуны на предыдущих этапах проигрывают своим соперникам, то нереально надеяться на то, что спортсмен, бегущий на последнем этапе, - это супермен, способный наверстать

ранее упущенное. Реально это или нереально, но на практике ситуация, складывающаяся в бизнесе, часто приводит к необходимости ускоренного выполнения стадии тестирования, поэтому всегда нужно быть готовым применить тестирование по приоритетам и другие планы на случай непредвиденных обстоятельств с целью смягчение последствий от овеществления рисков нарушения графика работ.


*Рис. 5.1. Обзор системных испытаний*

Основу тестирования составляет прогон тестов, включенных в план проведения испытаний. В результате прогона тестов обнаруживаются различные дефекты и выдаются сообщения об обнаруженных дефектах. При этом данные, обеспечивающие возможность отслеживания дефектов, вводятся в специальную базу данных, в которой хранится информация, предназначенная для отслеживания дефектов. В следующем разделе, "Обнаружение и отслеживание дефектов", эти аспекты тестирования рассматриваются более подробно.

Во время прогона тестов результаты тестирования должны передаваться другим подразделениям организации. Все, кто принимает участие в разработке, маркетинге или в руководстве проектом, должны знать, как продвигается тестирование, сколько и какие типы дефектов были обнаружены. По окончании тестирования должен быть подготовлен отчет, в котором подводятся итоги тестирования. В этом отчете должен использоваться набор критериев для оценки готовности программного продукта к выпуску. Например, если были выполнены все запланированные тесты, но еще остается какое-то допустимое число неустраненных дефектов, группа тестирования может утверждать, что программный продукт соответствует требованиям, предъявляемым к нему заказчиком, благодаря чему можно начинать поставки.

# Обнаружение и отслеживание дефектов

Цель тестирования заключается в нахождении дефектов. Однако недостаточно только отыскать дефекты. Об их наличии должным образом должны быть оповещены разработчики, чтобы они смогли эти дефекты устранить, а сведения о результатах выполненных ими исправлений доводятся до заинтересованных служб с тем, чтобы график разработки программного продукта не нарушался. Любые неэффективные действия при оповещении о дефектах, об их отслеживании, устранении и контроле приводят к потере времени.

В связи со сказанным выше, следующие два вида деятельности считаются основными при выявлении дефектов. Первым из них является прогон тестов в соответствии с планом проведения испытаний для выяснения, имеются ли какие-либо расхождения между ожидаемыми и фактическими результатами тестирования. Вторым видом деятельности является качественное документирование в системе отслеживания дефектов всех отклонений, обнаруженных во время тестирования. Хорошее качество документации важно по двум причинам — во-первых, разработчикам нужна достоверная и надежная информация для того, чтобы снять все порождаемые неисправностями проблемы, и, во-вторых, тестировщики должны иметь возможность воспроизвести проблемы, чтобы убедиться, что внесенные изменения устранили дефект. Система, применяемая для документирования и отслеживания дефектов, представляет собой важное инструментальное средство тестирования программного обеспечения. Несколько последующих разделов посвящаются изучению системы отслеживания дефектов, анализу состояний дефектов, изменению состояний дефектов и изучению других важных аспектов документирования и отслеживания дефектов.

## Определение состояний дефектов

В главе 1 отмечалось, что дефектом является ошибка в рабочем документе, таком как технические требования, план проведения испытаний или программный код. Дефект остается необнаруженным до тех пор, пока рабочий продукт не будет подвергнут пересмотру или не будет выполнен прогон программы, во время которого программа выдаст ожидаемые результаты.

После нахождения дефект подвергается анализу с целью определения, представляет ли он неисправность теста, ошибку в программном коде или дефект в проектных спецификациях. Если источником проблемы является тест, а не программный код, то тест должен быть исправлен, а обнаруженный дефект "проигнорирован". Проверяется также, насколько серьезен дефект: принадлежит ли рассматриваемый дефект к категории катастрофических, подлежащих обязательному устранению перед поставкой программного продукта? Является ли возникшая проблема настолько серьезной, что она существенно снижает функциональные возможности программного продукта, но при этом все еще остаются открытыми обходные пути, позволяющие продолжать процессы тестирования и разработки? Либо же эта проблема не настолько критична, что ее решение нельзя отложить до будущих выпусков программного продукта?

Программный код, содержащий обнаруженный дефект, должен быть направлен разработчикам для исправлений. Нужно внимательно следить за стадиями исправления, чтобы не потерять из виду сам дефект. Один из эффективных способов отеле-

живания дефекта состоит в четком определении явных состояний, которые может принимать дефект на пути от его обнаружения до завершающего тестирования. Каждое состояние должно быть недвусмысленным и обладать набором определений критериев входа и выхода из испытания.

В таблице 5.1 показан пример определения состояний дефекта. Названия состояний в каждой компании могут быть свои, возможны также дополнительные состояния, которые не указаны в этой таблице. В ней показан обобщенный набор названий и состояний, который, возможно, стоит использовать как отправную точку при создании системы отслеживания дефектов "с нуля".

**Таблица 5.1. Определение категории дефекта**

| Категория дефекта | Ответственность | Описание |
|---|---|---|
| Новый (new) | Группа тестирования | Дефект был обнаружен во время тестирования, этот факт был отражен в отчете, однако разработчик не получил распоряжения устранить проблему. |
| Исправить (fix) | Группа разработки | Разработчик получил распоряжение устранить проблему, и соответствующие работы ведутся. |
| Отложить (defer) | Группа анализа дефектов | Принято решение о том, чтобы отложить работы по устранению дефекта до последующих выпусков программного продукта. |
| Игнорировать (trash) | Группа анализа дефектов | Дефект был получен по ошибке, программный продукт работает в соответствии с проектным замыслом, возможны и другие причины того, что решено ничего не делать по устранению дефекта. |
| Исправлено (repair) | Группа разработки | Разработчик выявил и устранил основную причину дефекта и выполнил предварительное тестирование с целью убедиться в том, что проблема решена. |
| Исправлено и проверено (fix verified) | Группа тестирования | Специалист по тестированию проверил исправление, используя для этой цели ту же методику тестирования и конфигурацию средств тестирования, которые позволили первоначально обнаружить дефект. |

Отслеживание дефекта на протяжении его жизненного цикла, от его обнаружения до устранения и проверки правильности исправления, показывает, что он меняет свое состояние в соответствии с некоторым набором правил. На рис. 5.2 приведен пример изменения состояний дефекта. Дефект обнаруживается во время пересмотра или в процессе системных испытаний и попадает в систему отслеживания дефектов, будучи в состоянии "новый". Ему присваивается уровень серьезности, после чего он передается на рассмотрение в группу анализа дефектов. Группа анализа дефектов (известная в некоторых кругах как группа контроля за внесением изменений в техническую документацию) определяет, нужно ли исправлять дефект в рамках текущего выпуска или отложить это до будущих версий. Если группы разработчиков и тестировщиков провели дальнейший анализ уже после того, как дефект был зафиксирован, и пришли к заключению, что в программный код не содержит каких-либо проблем,

требующих его исправления, то группа анализа дефектов может принять решение не обращать на него внимания. Действие, предпринятое группой анализа дефектов, документируется, а сам дефект может находиться в состоянии "исправить", "отложить" или "игнорировать".

Если принято решение исправлять дефект, то эта работа поручается разработчику, который вносит соответствующие изменения в программный код и проводит тестирование, дабы удостовериться, что исправленный код работает правильно. Как только разработчик придет к выводу, что дефект устранен, он меняет состояние дефекта на "исправлено" и включает исправленный код в сборку, направляемую на системные испытания. Если тестировщик убеждается в том, что дефект устранен, то последний переходит в окончательное состояние "исправлено и проверено".

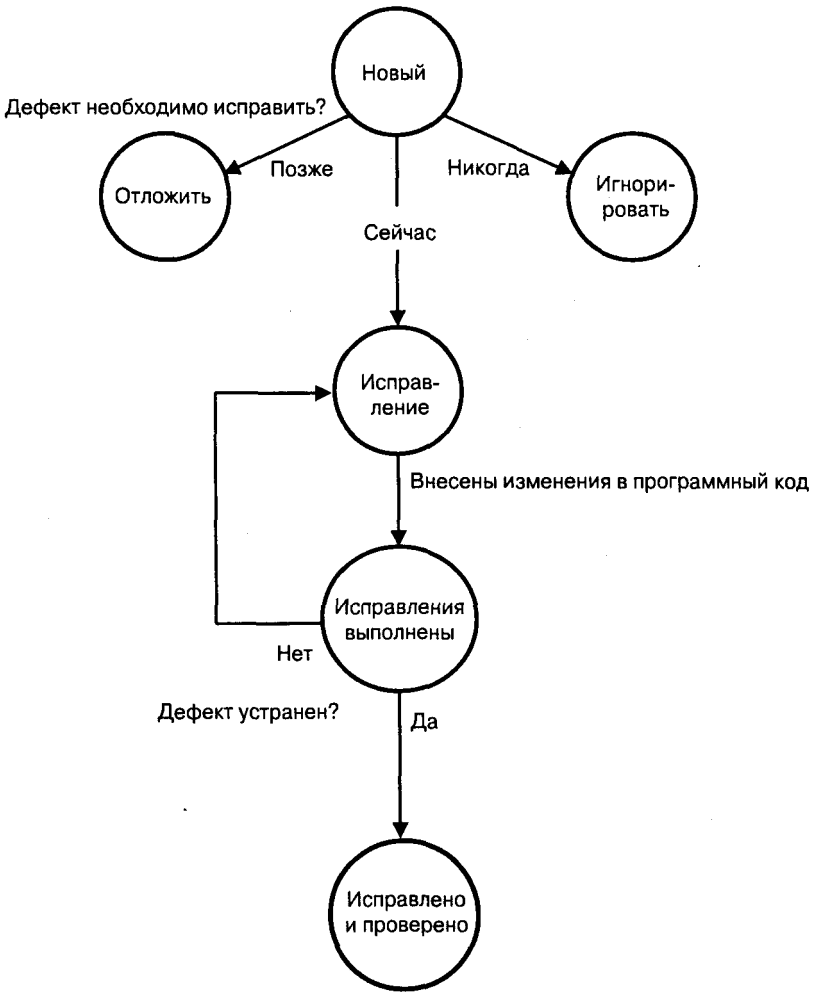



*Рис. 5.2. Как меняется состояние дефекта*

Если вы не обладаете опытом тестирования или работаете в компании, которая не использует мощных промышленный средств отслеживания дефектов, возможно, воз никнет впечатление, что методология отслеживания дефектов, в основе которой лежит состояние дефекта, чересчур избыточна и громоздка. Если вы занимаетесь разработкой программного продукта, выпуски которого редко когда содержат более 40 или 50 дефектов, то вы, скорее всего, правы. Однако в условиях разработки множества различных проектов количество дефектов, требующих отслеживания, достигае i многих сотен и даже тысяч. Если система отслеживания неэффективна, или если х<> тя бы всего лишь несколько заметных серьезных дефектов просочатся в поставляв мый программный продукт, руководству вашей испытательной лаборатории прел стоят довольно-таки неприятные объяснения с заказчиком.

По причине исключительной важности системы отслеживания дефектов для тес тирования программного обеспечения, следующие несколько разделов посвящаются обсуждению ее базовых характеристик.

## Базовые характеристики системы отслеживания дефектов

В момент самого первого столкновения с дефектом необходимо собрать о нем боль шую порцию информации, чтобы облегчить задачу его последующего устранения Пример информации о вновь обнаруженном дефекте, которую необходимо записать в системе отслеживания дефектов, приводится в таблице 5.2. В этой таблице приво дится список параметров с кратким описанием каждого параметра и причины необ ходимости его регистрации. Если для отслеживания дефектов применяются инстр\ ментальные средства управления базами данных, параметры, представленные в пер вом столбце, соответствуют полям базы данных.

**Таблица 5.2. Данные о дефекте, пребывающем в состоянии "Новый"**

| Характеристика | Описание | Обоснование |
|---|---|---|
| Идентификатор дефекта | Уникальный идентификатор дефекта, обычно это строка алфавитно-цифровых символов. | Позволяет отслеживать дефект в процессе изменения его состояний и логически связать с дефектом всю его обработку. |
| Состояние дефекта | Одно из допустимых состояний; в рассматриваемом случае таким состоянием является "новый". | Присвоить дефекту состояние "новый" в момент его обнаружения. |
| Кем обнаружен | Имя исполнителя, обнаружившего дефект. | Обеспечивает возможность задавать вопросы относительно дефекта. |
| Дата обнаружения дефекта | По меньшей мере, день/месяц/год; в этом поле может также быть указано время в часах и минутах. | Позволяет отслеживать хронологию событий, связанных с дефектом. Время в часах и минутах может оказаться полезным при отладке, если возникнет необходимость восстановить последовательность тестовых событий. |
| Обнаружен в сборке | Уникальный идентификатор сборки программного обеспечения, проходящего тестирование. | Этот показатель позволяет разработчикам выявить источник проблемы и дает возможность разработчику или тестировщику воспроизвести проблему. |

***Окончание табл. 5.2***

| Характеристика | Описание | Обоснование |
|---|---|---|
| Идентификатор | Уникальный идентификатор теста, во время прогона которого возникла проблема. Если эта проблема была обнаружена во время специализированного тестирования,это поле может быть помечено как NA (not applicable - не применимое). | Снабжает разработчика и тестировщика сведениями о том, прогон какого теста выполнять, чтобы воспроизвести проблему. Тест, который обнаруживает проблему, должен стать частью регрессивного тестирования, предназначенного для того, чтобы убедиться, что дальнейшие программные сборки не нужно подвергать регрессивному тестированию. |
| Краткое описание проблемы | Описание проблемы на концептуальном уровне. Обычно такое описание дефекта умещается в одной строке. | Это описание используется в сообщениях о статусе дефекта и в отчетных докладах и адресовано руководству проекта и членам других групп, стремящихся к пониманию проблемы во всех деталях. |
| Описание проблемы | Детальное описание симптомов проблемы и того, как она ограничивает функциональные возможности или производительность системы. | Это описание предназначено для тех, кому требуется детальное понимание проблемы, например, разработчику, который работает над устранением дефекта, или руководителю, распределяющему ресурсы. |
| Степень серьезности дефекта | Упорядочивание дефектов по степени серьезности последствий. Примеры:<br>**Катастрофический** - вызывает разрушение системы<br>**Крупный** - программный продукт не подлежит использованию<br>**Умеренный** - продукт можно использовать, однако имеют место некоторые неудобства для пользователя<br>**Незначительный** - пользователь не ощущает последствий<br>**Помеха** - можно устранить, когда позволит время. | Серьезность последствий часто определяет приоритеты усилий разработчиков при устранении дефектов и тестировщиков при проверке результатов этих исправлений. |
| Как воспроизвести проблему | Используется детально проработанная методика воспроизведения проблемы, включая выбор используемой конфигурации средств тестирования. Может оказаться достаточным сказать: "Выполнить прогон теста с идентификатором таким-то", если этот тест хорошо документирован. | Детально проработанная методика позволяет разработчикам воспроизвести дефект и предоставляет возможность разработчикам и тестировщикам проверять результаты устранения проблемы. |

Каждый дефект получает в базе данных отслеживания дефектов уникальный идентификатор, так называемый идентификатор дефекта. Идентификатор дефектов представляет собой ключ, который поддерживает запросы, поступающие в базу данных с таким расчетом, чтобы можно было получить информацию о его состоянии, которое можно рассматривать как показатель продвижения работ по его устранению. В базе данных отслеживания дефектов должны храниться сведения о том, кто обнаружил дефект, когда он был обнаружен, уровень серьезность дефекта, а также подробное описание проблем, которые он вызвал. Если дефект был обнаружен во время системных испытаний, должна быть записана информация, касающаяся теста, который выявил этот дефект. Для хорошо документированного тестового случая вполне достаточно запомнить только идентификатор теста. Если тест не задокументирован должным образом или если дефект был обнаружен в результате *специализированного (ad hoc)* тестирования, необходимо записать информацию о конфигурации средств тестирования и о специальных действиях, выполненных для обнаружения этого дефекта. Пример отчета о дефекте, который может использоваться для целей документирования, приводится далее в разделе "Как составлять сообщения о дефектах".

Данные о дефекте, попадающие в базу данных отслеживания дефектов, должны быть изучены группой анализа обнаруженных дефектов (или группой контроля за внесением изменений) с целью определения степени серьезности дефекта. На самом ли деле это дефект? Правильная ли степень серьезности была присвоена обнаруженному дефекту? Должен ли он быть устранен в текущем выпуске программного продукта? Информация, которая должна быть внесена в базу данных отслеживания дефектов по результатам работы группы анализа дефектов, показана в таблице 5.3. В результате анализа дефект переводится в одно из следующих трех состояний:

- **Исправить (fix)** — дефект переходит в это состояние, если группа анализа решит, что данных дефект должен быть исправлен в текущем выпуске.
- **Отложить (defer)** — дефект переводится в это состояние, если группа анализа решит отложить устранение этого дефекта до последующих выпусков программного продукта.
- **Игнорировать (trash)** — дефект переходит в это состояние, если группа анализа решила, что дефект был внесен по ошибке.

В зависимости от итогового состояния дефекта, в базу данных отслеживания дефектов вводятся различные данные. Например, если дефект двигается в направлении состояния "исправить", следует ввести имя исполнителя, ответственного за устранение дефекта. Если информация относительно дефекта готовится для передачи заказчикам, должен быть введен текст пояснительной записки по выпуску. Более подробная информация, касающаяся анализа дефектов, будет изложена в разделе "Анализ дефектов".

Как показано в таблице 5.4, окончательный набор данных нужно вводить в систему отслеживания дефектов во время исправления дефекта. Необходимо зафиксировать имя исполнителя, исправляющего дефект, дату исправления и идентификатор сборки, в которой производилось исправление. Должно быть дано подробное объяснение основной причины проблемы, чтобы группа разработки могла воспроизвести и устранить проблему.

**Таблица 5.3. Данные отслеживания дефекта в состояниях "исправить" (fix), "отложить" (defer) и "игнорировать" (trash)**

| Характеристика | Описание | Обоснование |
|---|---|---|
| Идентификатор дефекта (не изменяется) | Уникальный идентификатор дефекта | Позволяет отслеживать дефект в процессе изменения его состояний |
| Состояние дефекта | Состояния "исправить" отложить и "игнорировать" | Присвоить дефекту состояние "исправить" в момент его отправки разработчикам на исправление;присвоить дефекту состояние "отложить", если он будет устранен в следующем выпуске программного продукта; присвоить дефекту состояние "игнорировать", если его нельзя воспроизвести или если проблема за-_ключается не в дефекте, а в чем-то другом; дефект игнорируется также в случае, когда принято решение не исправлять его. |
| Кто исправил | Имя исполнителя, которому поручено устранить дефект. | Обеспечивает возможность задавать вопросы относительно исправления дефекта. |
| Кем отложен срок исправления | Имя ответственного лица, которое своим решением отодвинуло исправления дефекта на более поздний срок (оставить поле пустым, если исправление дефекта не отложено). | Обеспечивает возможность задавать вопросы относительно решения отложить исправление дефекта на более поздний срок. |
| Кто проигнорировал дефект | Имя ответственного лица, которое приняло решение не исправлять дефект (оставить поле пустым, если исправление дефекта не откладывалось). | Обеспечивает возможность задавать вопросы относительно решения не исправлять дефект. |
| Дата начала исправления | День, месяц и год, когда устранение дефекта было поручено разработчику, ответственному за исправления. | Позволяет отслеживать хронологию событий, связанных с дефектом. |
| Дата, когда исправление было отложено | День, месяц и год, когда исправление было отложено. | Позволяет отслеживать хронологию событий, связанных с дефектом. |
| Дата, когда было принято решение не делать исправления. | День, месяц и год, когда было принято решение не делать исправления. | Позволяет отслеживать хронологию событий, связанных с дефектом. |
| Нужна пояснительная записка к выпуску (Y/N)? | Введите 'Y', если нужна пояснительная записка к выпуску, и "N", если такая пояснительная записка не нужна. | Помечает дефект флагами таким образом, чтобы можно было составить список дефектов, для которых требуются пояснительные записки к выпуску. |

***Окончание табл. 5.3***

| Характеристика | Описание | Обоснование |
|---|---|---|
| Информация о выпуске программного продукта | Описание дефекта, которое будет передано заказчику как составная часть пояснительных записок, прилагаемых к выпуску (необходимы только в тех случаях, когда дефекты не исправлены в конкретном выпуске). | |

**Таблица 5.4. Данные отслеживания дефекта в состояниях "исправлено" (repaired) и "исправлено и проверено" (fix verified)**

| Характеристика | Описание | Обоснование |
|---|---|---|
| Идентификатор дефекта (не изменяется) | Уникальный идентификатор дефекта | Позволяет отслеживать дефект в процессе изменения его состояний |
| Состояние дефекта | Состояния "исправлено", "исправлено и проверено". | Присвоить дефекту состояние "исправлено", когда исправление внесено в основание кода; присвоить дефекту состояние "исправлено и проверено", когда специалист по тестированию подтвердил,что исправленный код работает правильно. |
| Кто вносил исправления | Имя исполнителя, который устранял дефект. | Обеспечивает возможность задавать вопросы относительно исправления дефекта. |
| Кто проверял исправления | Имя исполнителя, который проверял исправления. | Обеспечивает возможность задавать вопросы относительно проверки исправления дефекта. |
| Дата, когда исправление было сделано | День, месяц и год, когда исправление было внесено в код. | Позволяет отслеживать хронологию событий, связанных с дефектом |
| Дата, когда исправление было проверено | День, месяц и год, когда исправление было проверено. | Позволяет отслеживать хронологию событий, связанных с дефектом |
| Исправление внесено в сборку | Уникальный идентификатор сборки, в которую будет внесено исправление. | Позволяет тестировщику знать, какую сборку следует использовать, чтобы проверить исправление; если исправление дефекта отложено до последующих выпусков программного продукта, то это поле показывает, в рамках какого выпуска планируется внести исправление. |
| Описание исправления | Объяснение основной причины проблемы и как она была исправлена. | Препятствует возникновению подобных проблем в будущем; помогает разработчику и тестировщику выявлять другие дефекты, имеющие отношение к первоначальной проблеме. |

В таблицах с 5.1 по 5.4 дано краткое описание некоторых основных свойств, которыми должна обладать система отслеживания дефектов. Можно построить собственное инструментальное средство отслеживания дефектов, тем не менее, существуют различные серийно выпускаемые инструментальные средства, позволяющие сэкономить массу времени и усилий, которые пришлось бы потратить на разработку эффективной системы отслеживания дефектов. Набор возможностей и удобство и простота использования вашей системы отслеживания дефектов существенно влияет на эффективность применяемой организации тестирования, поэтому выбор системы отслеживания дефектов следует производить максимально аккуратно.

В следующем разделе дано описание формата сообщения о дефекте, который хорошо вписывается в только что приведенную общую схему.

## Как составлять сообщения о дефектах

Каждый раз, когда обнаруживается новый дефект, должно составляться письменное сообщение об этом дефекте. Если информация о дефекте не будет получена достаточно оперативно, могут возникнуть трудности при восстановлении обстоятельств, которые привели к возникновению проблемы. Это тем более справедливо, если применяются специализированные виды тестирования. Если выполняется прогон документированного тестового случая, то его исход может быть обозначен как неудачный, а отличия фактических результатов тестов от ожидаемых могут быть зарегистрированы как составная часть прогона теста. Как только специализированный вид тестирования обнаруживает проблему или как только прогон теста завершен, должно быть составлено сообщение о дефекте.

Пример шаблона сообщения о дефекте показан на рис. 5.3. Этот шаблон может быть представлен в форме печатного бланка, который вручную заполняется тестировщиком, в виде шаблона ввода данных в Web-формате или как часть базы данных системы отслеживания дефектов. Такой шаблон сообщения о дефекте представляет собой средство сбора тестовых данных, которые отображаются в таблице 5.2.

От умения специалистов по тестированию составить толковое сообщение о дефекте в значительной мере зависит успех их деятельности. Часто время, затрачиваемое тестировщиками на изучение сообщений о дефектах, которые были составлены их организациями во время разработки аналогичных проектов, приносит большую пользу. В идеальном случае имеется набор "золотых примеров", который дает возможность тестировщикам, не имеющим достаточного опыта, получить понимание того, что такое высококачественное сообщение о дефекте и какие факторы делают такое сообщение неэффективным.

Вообще говоря, сообщение о дефекте считается плохо составленным или неэффективным, если:

- Оно составлено по ошибке — дефект, описанный в сообщении, не существует.
- Возникшая проблема представлена в описании нечетко или неоднозначно — проблема, возможно, и существует, но в то же время она описана настолько плохо, что поведение системы не ясно.
- Оно не содержит информации, необходимой разработчику для воссоздания проблемы — признаки проблемы описаны, но при этом отсутствуют пояснения к действиям, которые послужили причиной возникновения проблемы.

| Сообщение о дефекте |
| --- |
| Идентификатор дефекта: |
| Кем обнаружен: |
| Дата обнаружения: |
| Обнаружен в сборке: |
| Степень серьезности дефекта (катастрофический, крупный, незначительный): |
| Идентификатор теста: |
| Краткое описание проблемы: |
| Подробное описание проблемы: |
| Как воспроизвести проблему:<br>(Если был использован недокументированный тест, подробно опишите методику тестирования. При необходимости дайте схему конфигурации средств тестирования.) |

***Рис. 5.3. Шаблон сообщения об ошибке***

Правильно составленное сообщение о дефекте обладает совершенно другими свойствами:

- Оно описывает фактический дефект программного продукта.
- В нем четко описаны признаки проблемы через отклонения поведения системы от нормы.
- .Оно содержит процедуру пошагового воспроизведения проблемы.

## Анализ обнаруженных дефектов

Если бы процесс обнаружения и исправления дефектов был совершенным, то необходимость в анализе дефектов отпала бы сама по себе. К сожалению, человеческий фактор вступает в действие уже на стадии системных испытаний, впрочем, как и на других этапах цикла разработки. Ошибки могут неожиданно обнаружиться в тестовых случаях, в конфигурациях средств тестирования или в интерпретации данных тестирования. Когда обнаружен дефект, необходимо проследить за тем, чтобы своевременно был назначен исполнитель, ответственный за его устранение, что обеспечит исправление дефекта и проверку результатов этого исправления.

Один из способов проверки результатов тестирования и предотвращения возникновения ошибок при отслеживании дефектов предусматривает еженедельный анализ дефектов, обнаруженных во время системных испытаний. Назначение анализа дефектов состоит в том, чтобы:

- Выявить все очень серьезные дефекты, требующие немедленного внимания
- Выявить все дефекты, которые требуют более глубокого исследования или не могут быть воспроизведены.
- Определить изменения состояний дефектов.

Анализ дефектов проводится не так, как пересмотр программных кодов или других рабочих продуктов. В большинстве случаев экономически нецелесообразно вести учет времени, затраченного исполнителями на подготовку к анализу, к тому же роли участников не расписаны так же четко, как в случае перепроверки программных кодов и тестовых случаев. Тем не менее, полезно выработать подходящую процедуру проведения анализа дефектов. Часто в испытательных лабораториях совещания, посвященные анализу дефектов, проводят ведущие инженеры, при этом дефекты, тре; бующие анализа, представлены в рамках отчетного доклада, который можно рассматривать как высокоуровневую информацию о дефектах. Пример формата отчетного доклада, посвященного дефектам, показан на рис. 5.4.

| **Анализ дефектов проекта** | | | | | |
|---|---|---|---|---|---|
| Дата анализа: день/месяц/год | | | | | |
| Участники: | | | | | |
| **Идентификатор дефекта** | **Состояние** | **Серьезность дефекта** | **Дата обнаружения** | **Краткое описание дефекта** | **Действие** |
| B1233 | Новый | Катастрофический | День/месяц/год | Нарушение функций резервирования/восстановления - невозможен поиск данных после резервирования. | Исправить |
| B1234 | Новый | Крупный | День/месяц/год | Нельзя открыть резервный экран из меню; обходной путь - использование командной строки. | Исправить |
| B1235 | Новый | Незначительный | День/месяц/год | Проблема удобства использования - не работает вызов функции поиска через клавишу ускоренного доступа. | Отложить |
| | | | | | |
| | | | | | |
| Примечания:<br>B1235 — эта проблема требует изменений в архитектуре программного продукта, в силу чего ее решение откладывается до следующей версии. | | | | | |

*Рис. 5.4. Пример отчета по результатам анализа дефектов*

В примере сообщения на рис. 5.4 приводится список всех дефектов с указанием их идентификаторов, а также содержится информация об их состоянии, степени серьезности и дате обнаружения. Дается краткое описание проблем, но если потребуется подробное описание конкретного дефекта, возникает потребность в первоначальном сообщении о дефекте. Одним из ключевых полей в отчете является столбец "Действие". В поле предполагаемого действия можно определить какое-то значение еще до того, как будет выполнен анализ, однако основной результат совещания заключается в определении конкретных состояний, которые должны быть указаны в этом столбце.

В сообщении, приведенном на рис. 5.4, резервируется место для ввода данных анализа, для перечисления участников, а также поле "Примечания", которое может использоваться для документирования обоснования решений, принятых на совещании. Если по результатам анализа все поля заполняются корректно, итоговый отчет послужит источником для того, чтобы расписать совещание по минутам.

# Прогон тестов

До сих пор основное внимание уделялось дефектам, ибо обнаружение дефектов является основной причиной для прогона тестов. Далее мы сосредоточимся на изучении самого процесса тестирования, следуя обзорной диаграмме, которая показана на рис. 5.1.

## Вход в системные испытания

При передаче группе тестирования новой сборки применяется некоторый набор критериев входа в испытания. Перед началом системных испытаний должен быть разработан и утвержден план проведения испытаний и тестовые случаи. Группа разработки должна закончить тестирование модулей и проверку взаимодействия и функционирования компонентов этой сборки. Они должны исправить все катастрофические дефекты, обнаруженные во время испытаний, и только затем передавать сборку группе тестирования.

Первый тест, прогоняемый на новой сборке, — это обычно тест "на герметичность", который служит для проверки возможности установки программы на целевой платформе, возможности успешного обновления и определения факта работоспособности, по меньшей мере, базовых функций. Причина проведения теста на герметичность довольно проста: если программа не работает на целевой платформе, ее просто невозможно тестировать.

К сожалению, нередки случаи, когда новая сборка не проходит испытания на герметичность, особенно если перед ее передачей тестировщикам проверка взаимодействия и функционирования компонентов была выполнена в недостаточных объемах. Разработчики могли настолько сосредоточиться на том, чтобы заданные функции успешно работали в среде разработки, что до проблем системного уровня, таким как, скажем, установка программы и ее работа в среде заказчика, просто не доходили руки.

## Циклы тестирования

Вход в системное тестирование должно означать начало совместных работ при участии групп разработчиков и тестировщиков. Тестировщики выполняют прогон запланированных тестов, выявляют дефекты и пишут сообщения о дефектах. Разработчики читают сообщения о дефектах, воспроизводят проблемы и исправляют программный код. Вопрос заключается вот в чем: как исправления передаются в группу тестирования?

Ответ на этот вопрос зависит от цикла создания программного обеспечения. Разработчики регистрируют проделанные исправления в системе управления конфигурациями в рамках подготовки к периодическим построениям сборок. Построение сборки обычно выполняется на базе ежедневного или еженедельного цикла, после чего блок передается группе тестирования в соответствии с их потребностями.

Возникают различные вопросы относительно того, как часто группа тестирования может принимать новые сборки. Если сборки приходят слишком часто, устойчивость среды тестирования может оказаться нарушенной — потребуется время на установку новых сборок, прогон тестов на герметичность и проверку того, что дефекты, которые были обнаружены в предыдущей версии сборки, исправлены. Если "смесь сборок" будет излишне "пестрой", то попросту не хватит времени на прогон достаточного количества тестов, дабы обеспечить эффективное обнаружение дефектов в соответствии с выбранной методологией тестирования.

С другой стороны, если сборки поступают слишком медленно, то и в этом случае нельзя достичь высокой эффективности обнаружения дефектов. Довольно часто изменения, вносимые в программный код, приводят к тому, что обнаруживаются новые дефекты, либо выявляются дефекты, которые и раньше были в сборке, но были замаскированы теперь уже устраненными дефектами. Это значит, что включать новые сборки в цикл тестирования нужно достаточно часто, чтобы можно было обнаружить следующую партию дефектов.

То, как часто доводится принимать новые сборки от разработчиков, зависит от нескольких факторов, в том числе и от сложности программного обеспечения, от численности штата тестировщиков, от процентного отношения автоматизированных тестов к общему числу тестов. Например, если можно прогнать все тесты в течение трех дней, вероятно, имеет смысл принимать новые сборки каждые четыре-пять дней. В этом случае три дня можно отвести на прогон тестов, а день или около того — на проверку исправлений дефектов, на прогон специализированных тестов в некоторых многообещающих областях, на подготовку отчетов об устранении дефектов и на регулярный анализ дефектов.

В идеальном случае *цикл тестирования (test cycle)* состоит из прогона полного набора тестов на некоторой сборке программного обеспечения. План проведения испытаний обычно предусматривает выполнение некоторого количества циклов тестирования, причем на каждый цикл затрачивается определенное время. Например, этот план может предусматривать выполнение трех или четырех циклов длительностью в одну или две недели каждый.

На практике тестирование не всегда проводится с соблюдением плана проведения испытаний. Первый цикл тестирования может продолжаться одну-две недели, однако может потребоваться большое число сборок, чтобы тестирование оказалось достаточно устойчивым и можно было прогнать большую часть тестов. Когда очередная

новая сборка поступает на тестирование, принимается решение, нужно ли выполнить повторный прогон всех тестов с самого начали или продолжать тестирование в прежней последовательности. Обычно более эффективной оказывается выборочная проверка новой сборки, которая дает возможность убедиться в том, что ее можно остановить и запустить в работу, а обнаруженные ранее дефекты уже устранены. После этого тестирование продолжается без повторного прогона всех тестов. Подходящим моментом для очередного запуска тестирования может служить начало нового тестового цикла. Это означает, что может произойти так, что на промежуточной сборке не будет выполнен полный набор тестов. Исключением из этой стратегии является завершающая сборка программного обеспечения, так называемая "золотая сборка", на которой прогоняются все тесты.

Один из способов отслеживания тестов, прогон которых выполнен на заданной сборке, предусматривает ведение списка прогона тестов на сборке для каждой сборки тестируемого программного обеспечения. Пример списка прогона тестов на сборке показан на рис. 5.5. В этом списке в качестве заголовка указан идентификатор сборки и дата прогона теста; список прогона тестов, по существу, представляет собой список того, "что нужно сделать" каждому специалисту по тестированию. Если планируется применение испытательных комплексов или рабочих станций коллективного пользования, этот список позволит избежать конфликтов, если в нем для каждого специалиста по тестированию будет указываться испытательный комплекс, на котором должны запускаться тесты. Если же возникнут конфликты требований на использование испытательного комплекса, то в таких случаях может потребоваться разработка рабочего графика для комплексов. В списке прогона тестов на сборке один столбец резервируется под краткое изложение результатов тестирования.

| **Список прогона тестов на сборке** | | | | |
|---|---|---|---|---|
| Идентификатор сборки: | | | | |
| Дата начала тестирования: | | | | |
| **Номер теста** | **Идентификатор теста** | **Имя тестировщика** | **Конфигурация средств тестирования** | **Результат (P/F/NR)** |
| **1.** | **A10** | JimD. | 1A на системе 1 | Pass (тест прошел) |
| **2.** | **A11** | Jim D. | 1A на системе 1 | Pass (тест прошел) |
| **3.** | **B11** | BobZ. | 2B на системе 2 | Fail (тест не прошел) |
| **4.** | **B11** | BobZ. | 2B на системе 2 | Not run (тест не выполнялся) |
| | | | | |
| | | | | |

*Рис. 5.5. Пример списка прогона тестов на сборке*

Список прогона тестов на сборке может оказаться исключительно полезным при частой передаче сборок в группу тестирования. На основе информации о тестах, прогон которых выполнялся на предыдущих сборках, список можно составить так, чтобы обеспечить эффективное покрытие тестами свойств программного продукта на некоторой последовательности сборок. Список прогона тестов на сборке помогает также приспосабливаться к ситуации, когда в заданной сборке присутствуют дефекты, которые не позволяют обеспечить покрытие конкретной области программного кода. Как только блокирующий дефект будет исправлен, можно сосредоточить свое внимание на этой области и обеспечить высокую степень покрытия.

## Регистрация результатов тестирования

Когда тестировщик получает задание прогнать некоторую последовательность тестов на конкретной сборке, он должен следовать пошаговой методике тестирования, оговоренной в тестовом случае, и протоколировать результаты тестирования. Как видно из примера тестового случая, приведенного в главе 4 (рис. 4.2), в нем отводится специальное место, куда тестировщик заносит результаты выполнения каждого шага, равно как и исход испытания (прошел, не прошел). Кроме того, выделяется также и место, куда заносится идентификатор любого дефекта, обнаруженного во время испытаний. Один из способов регистрации результатов прогона тестов предусматривает использование формата тестового случая, который показан на рис. 4.2.

Другой способ заключается в применении журнала испытаний, изображенного на рис. 5.6. Этот журнал испытаний удобен для сведения воедино результатов всех тестов, выполненных одним специалистом по тестированию на одной сборке. В нем также есть место для регистрации общего результата "прошел/не прошел/тест не выполнялся", а также для идентификатора любого дефекта, обнаруженного во время прогона теста. Предусматривается также место для комментариев — обычно в нем протоколируется информация, которая может впоследствии оказаться полезной при воспроизведении проблемы или для понимания условий, в которых проблема наблюдалась.

Независимо от того, какой метод будет использоваться для регистрации результатов тестирования, очень важно, чтобы результат каждого теста был зарегистрирован и сохранен в безопасном архиве. Заархивированные результаты тестирования могут позже понадобиться в силу различных причин: подтверждение факта проведения тестирования, поддержка воспроизведения проблемы или хронологические записи, обосновывающие трудозатраты на разработку программного продукта.

Шаблоны списка прогона тестов на сборке, тестовых случаев и журналов испытаний приводятся в данной книге в форме текстовых документов, тем не менее, их можно реализовать и как часть системы ввода данных через Web либо инструментальных средств организации испытаний. Использование автоматизированных средств для сохранения тестовых случаев, прогона тестов и генерации отчетов по испытаниям может существенно повысить качество построения эффективной стратегии системных испытаний.

| Журнал испытаний | | | | |
|---|---|---|---|---|
| Дата: день/месяц/год | | | | |
| Цикл тестирования: 1 | | | | |
| Идентификатор сборки: B020202 | | | | |
| Конфигурация средств тестирования: 1A | | | | |
| Тестирование выполнял: Jim D. | | | | |
| **Тест №** | **Идентификатор теста** | **Исход Pass/Fail/Not run** | **Идентификатор дефекта** | **Комментарий** |
| **1.** | **A10** | **P** | | |
| **2.** | A11 | **P** | | |
| **3.** | **A12** | **P** | | |
| **4.** | **A13** | **F** | XY1233 | Эта проблема характерна для конфигурации средств тестирования. |
| **5.** | **A14** | **P** | | |
| **6.** | **A15** | **F** | XY1234 | |
| **7.** | **B1** | **NR** | | Тестовый набор заблокирован дефектом XY1234. |
| **8.** | **B2** | **NR** | | Заблокирован |
| **9.** | **B3** | **NR** | | Заблокирован |
| **10.** | **B4** | **NR** | | Заблокирован |

***Рис. 5.6. Журнал испытаний***

# Составление отчетов по результатам тестирования

Обычно требуются два вида отчетов по результатам тестирования. Первый из них — это регулярный доклад о ходе тестирования, который часто делается на еженедельных совещаниях. На таких совещаниях представители всех коллективов, ответственных за успех проекта, отчитываются о проделанной работе. Доклад о ходе тестирования обычно состоит из отчета о ходе работ по тестированию и отчетного доклада об обнаруженных дефектах. Доклад о ходе работ должен дать ответ на вопрос "насколько далеко мы продвинулись в тестировании?". Отчетный доклад об обнаруженных дефектах предназначен для выяснения степени успешности тестовых мероприятий в контексте обнаружения дефектов. Одна из его целей состоит в получении текущей оценки качества тестируемого программного продукта. Эти отчеты рассматриваются в двух следующих разделах.

Второй тип доклада по результатам тестирования представляет собой формальную сводку результатов тестирования, которая составляется по окончании системных испытаний. Доклад второго типа предназначен для документирования результатов тестирования с тем, чтобы в будущем можно было получить ответ на вопросы о том, что подвергалось тестированию, какие были получены результаты, и какого мнения придерживалась группа тестирования относительно готовности продукта к выпуску. Этот отчет рассматривается далее в разделе "Отчетный доклад".

## Отчет о ходе работ по тестированию

Отчет, показанный на рис. 5.7, дает представление о ходе работ по тестированию. Из информации о дате, идентификаторе сборки и цикле тестирования видно, какой программный продукт проходит тестирование и приблизительно насколько проводимое тестирование отклоняется от плана. Последний столбец отчета, % Run ("процент прогонов"), есть мера того, сколько прогонов тестов было выполнено относительно общего количества запланированных тестов. Число тестов, прогон которых завершился неудачно, указывается в столбце "# Fail" (тест не прошел) — этот показатель дает приближенные данные о том, сколько дефектов будет зафиксировано в проверяемом программном продукте. Столбец "# Not Run" (тест не выполнялся) показывает, сколько тестов заблокировано по причине неустраненных дефектов или других проблем. Общее количество тестов приводится в столбце "# Tests", а количество успешно пройденных тестов - в столбце "# Pass".

| **Отчет о ходе работ по тестированию** | | | | | |
|---|---|---|---|---|---|
| Дата составления отчета: день/месяц/год | | | | | |
| Идентификатор сборки: B123 | | | | | |
| Цикл тестирования: 2/3 | | | | | |
| Дата начала тестовых работ: месяц / день / год | | | | | |
| **Тестовый набор** | **# Tests** | **#Pass** | **# Fail** | **# Not Run** | **% Run** |
| Ввод данных | **20** | **12** | **2** | **6** | **70%** |
| Редактирование | **15** | **10** | **4** | **1** | **93%** |
| Резервирование и восстановление | **25** | **12** | **6** | **7** | **72%** |
| Обмен данными | **18** | **8** | **8** | **2** | **89%** |
| Итого | **78** | **42** | **20** | **16** | **79%** |

***Рис. 5.7. Пример отчета о ходе работ по тестированию***

## Отчет об устранении дефектов

Другим видом анализа положения дел в тестировании является отчет об устранении дефектов, пример которого представлен на рис. 5.8. В этом отчете показано общее число неустраненных дефектов как функция от времени в неделях. Временная ось

часто градуируется в датах, а не в "неделя 1", "неделя 2" и т.д. Количество дефектов заданной серьезности (катастрофические, крупные и незначительные) в этом отчете выглядит как столбик соответствующей высоты в выбранном масштабе, а также указывается явно в числовой форме. Например, на третьей и четвертой неделе тестирования в программном продукте было зафиксировано 4 катастрофических дефекта, а на второй неделе —32 незначительных дефекта.

Общее количество неустраненных дефектов есть показатель качества программного продукта. Вполне понятно, что большое количество дефектов высокого уровня серьезности не говорит в пользу качества программного продукта. Окончательным определителем качества программного продукта является то, насколько он устраивает заказчика. Разумеется, заказчик вряд ли будет доволен, если в предоставленной ему версии продукта присутствует слишком много дефектов. Иногда имеются смягчающие вину обстоятельства, когда наиболее важной является скорость поставки продукта, а заказчик согласен даже на версию с дефектами, лишь бы что-то работало.

По мере того как работы по тестированию приближаются к завершению, есть все основания ожидать, что число неустраненных дефектов будет уменьшаться, поскольку разработчики исправляют их, а тестировщики проверяют, насколько правильно были выполнены исправления. Степень серьезности дефектов по мере продвижения тестирования также имеет тенденцию к снижению. Пример, представленный на рис. 5.8, необычен тем, что количество неустраненных катастрофических и крупных дефектов остается практически на одном уровне, в то время как число незначительных дефектов существенно снижается. Более типичным случаем является быстрое устранение дефектов с высоким уровнем серьезности, в то время как менее серьезные дефекты остаются не устраненными в течение более длительного времени; возможно даже, что их устранение откладывается до будущих выпусков.


*Рис. 5.8. Пример отчета об устранении дефектов*

Другой отчет, который часто используется при анализа состояния проекта, — отчет об анализе дефектов, представленный в таблице 5.5. Диаграмма на рис. 5.4 достаточно точно характеризует ход тестовых работ и служит показателем качества программного продукта, однако все решения относительно того, исправления каких дефектов будут включаться в выпуск, и в каком порядке дефекты будут устраняться, требуют более подробной информации о неустраненных дефектах. В некоторых компаниях функции анализа проекта и анализа дефектов объединены, причем оба вида анализа регулярно ставятся на повестку дня заседаний группы контроля за внесением изменений CCB (change control board). Как правило, в состав группы CCB входят представители руководства, отдела маркетинга, организаций, которые занимаются разработкой и тестированием.

## Отчетный доклад

В отчетном докладе должны быть отражены следующие моменты:

- Какой программный продукт тестировался
- Насколько фактические работы по тестированию отклонились от плана проведения испытаний
- Насколько график работ и трудозатраты отличаются от соответствующих показателей в плане проведения испытаний
- Какие дефекты были обнаружены
- Какие дефекты остались неустраненными по завершении тестовых работ и что с ними делать дальше.

Отчетный доклад по результатам тестирования не обязательно должен содержать результаты прогона каждого теста, в то же время он может ссылаться на такие результаты, если они где-то накапливаются и архивируются. Сбор и архивирование всех тестовых результатов — это одно из преимуществ, которое дает использование серийных инструментальных средств управления тестированием. Если это инструментальное средство сводит результаты тестирования в единый документ, то на этот документ можно сослаться в отчетном докладе.

Один из способов указать, какой продукт подвергается тестированию, предусматривает использование набора окончательных вариантов журналов испытаний (рис. 5.6) на каждом цикле испытаний. Этот способ позволяет показывать результаты прогона каждого теста без использования пошагового отчета наподобие приведенного на рис. 4.2 в главе 4. Другой полезный отчет можно получить за счет компиляции полного набора данных о ходе работ по тестированию (см. рис. 5.7).

Важно понять, насколько фактический процесс тестирования отклоняется от плана проведения испытаний, поскольку план проведения испытаний представляет собой утвержденное определение объемов тестирования и соглашение о том, сколько сотрудников принимают участие в этих работах, и сколько времени будет затрачено на выполнение работ. Особенно важно отметить, какие области не были охвачены тестированием в полном объеме, поскольку эти области являются источниками рисков снижения качества программного продукта и источниками распространения дефектов в различные части продукта. Если тестирование проходит в соответствии с планом, этот факт можно просто отметить в отчетном докладе; если имеют место

етные отклонения от плана, необходимо отметить их в специальном заявлении. информация полезна при учете затраченных ресурсов, и в то же самое время ит исходными данными для дальнейшего планирования. Если производится сбор ^^ дставительных статистических данных для сравнения запланированных и фактиS ^ n K H x трудозатрат, то она благоприятствует возможности повышения точности бу^™——их оценок.

Несмотря на то что база данных отслеживания дефектов должна содержать полный набор данных о дефектах, обнаруженных во время тестирования программного продукта, сводку дефектов, обнаруженных во время системных испытаний, целесообразно включить в отчетный доклад. В таком случае всем сторонам, заинтересованным в успехе разработки, не потребуется обращаться в базу данных дефектов за сведениями о ходе работ по тестированию и о качестве программного продукта. Окончательная версия отчета по результатам анализа дефектов (рис. 5.4), в котором показаны все обнаруженные дефекты и их окончательные состояния, а также окончательная редакция отчета о категориях неустраненных дефектов (рис. 5.8) являются полезными дополнениями отчетного доклада о тестировании.

Отчетный доклад о результатах тестирования должен содержать список всех неустраненных дефектов с указанием того, будут ли они исправлены в более поздних выпусках или же их исправление откладывается на неопределенное время. Дальнейшие выпуски программного продукта все еще будут содержать неустраненные дефекты до тех пор, пока они не будут исправлены. В силу этого обстоятельства желательно вести за такими дефектами наблюдение, чтобы не тратить дополнительные усилия на их выявление в будущих выпусках продукта. В плане проведения испытаний будущих выпусков должны присутствовать ссылки на список неустраненных дефектов; это позволит тестировщикам понять, что их ожидает во время тестирования нового выпуска программного продукта.

Пример отчетного доклада о результатах тестирования, котором реализованы все идеи из этого раздела, включен в третью часть книги.

## Критерий выхода из испытаний и готовность выпуска программного продукта

Существует множество способов определить подходящий момент для останова испытаний, одни из них простые, другие же — очень сложные. Вот некоторые из условий, которые используются для принятия решения о прекращении испытаний:

- Время, отведенное на тестирование, истекло. Если зафиксирован некоторый предельный срок, неизбежно наступает день, когда вы просто прекращаете все работы, связанные с тестированием, и задаете себе вопрос: "Насколько плох программный продукт?". Любой другой метод останова придает гораздо большую уверенность в качестве поставляемого программного продукта.
- Завершены все запланированные циклы тестирования. Если план проведения испытаний предусматривает три цикла, то в конце третьего цикла вы, по-видимому, зададите тот же вопрос: "Насколько плох программный продукт?" Если план проведения испытаний выполнен, то тестовое покрытие, скорее

всего, будет лучше, чем в случае, когда просто не хватило времени для доведения тестирования до конца.

- **Профиль дефектов соответствует критерию выхода из испытаний.** Если в результате выполнения алгоритма SWEEP (Software Error Estimation Program — Программа оценки ошибочности программного продукта) становится ясно, что предел, т.е. количество неустраненных ошибок на тысячу строк программного кода, достигнут, появляются все основания прекратить тестирование. (Подробную информацию по алгоритму SWEEP можно найти в главе 11.) В подобной ситуации опять-таки придется задаться вопросом: "Насколько плох программный продукт?"

Принимая решение прекратить работы по тестированию, следует учесть несколь; ко факторов, которые играют важную роль при оценке готовности программного продукта к поставкам. Обычным набором факторов, которые определяют готовность программного продукта к поставкам, являются:

- Количество катастрофических дефектов, обнаруженных в процессе тестирования, которые остаются неисправленными
- Общее количество дефектов, которые не были исправлены
- Процентное отношение количества тестов, которые завершились успешным исходом, к числу всех запланированных тестов
- Количество тестов, прогон которых не может быть выполнен, поскольку они заблокированы дефектами.

С целью выработки оценки готовности выпуска отлаженного программного продукта проводятся специальные совещания. В некоторых организациях оценку готовности определяет группа тестирования; в других организациях совещание проводит руководитель проекта либо руководитель разработки. В любом случае в задачу группы тестирования входит представление результатов тестирования и выработка рекомендаций относительно готовности продукта к поставкам. Хотя оценка готовности может проводиться формально, подобно тому, как выполняется инспекция программного кода, в любом случае потребуется зафиксировать имена участников и принятое ими решение, касающееся поставок программного продукта. Чтобы оценка готовности содержала результаты и рекомендации организации, проводившей тестирование, эта оценка должна содержать ссылку на отчетный доклад по результатам тестирования.

# Что дальше

В этой главе мы обсуждали вопросы отслеживания дефектов, проведения системных испытаний и составления отчетов по результатам тестирования. Рассматривались следующие ключевые моменты:

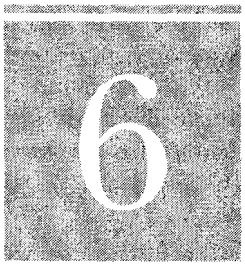
- Обзор системных испытаний
- Обнаружение и отслеживание дефектов
- Определение состояния дефектов
- Основные особенности отслеживания дефектов

- Составление сообщений о дефектах
- Анализ дефектов
- Прогон системных тестов
- Вход в системные испытания
- Циклы тестирования
- Регистрация результатов прогона тестов
- Отчетность по результатам тестирования
- Отчет о ходе выполнения тестовых работ
- Отчет по результатам тестирования
- Критерии выхода из испытаний и оценка готовности

Необходимыми условиями для проведения системных испытаний являются план проведения испытаний в его окончательном виде, набор готовых к прогону тестовых случаев и отлаженный испытательный комплекс в требуемой конфигурации. Результат выполнения тестовых работ представляет собой совокупность результатов тестирования, которые позволяют руководству проекта и коллективу разработчиков выработать оценку готовности программного продукта к выпуску.

В первых пяти главах книги даны определения множества процессов, которые охватывают тестирование программного обеспечения от этапа выявления требований до завершения системных испытаний. Если вы впервые сталкиваетесь с тестированием или пытаетесь открыть новую испытательную лабораторию, трудно будет реализовать все эти процессы сразу; их необходимо разворачивать постепенно, как часть долгосрочной программы совершенствования процессов. В следующей главе будет показано, как построить интегрированный процесс тестирования, воспользовавшись подходом долгосрочного совершенствования.

# Вопросы объединения процессов тестирования и кадрового обеспечения



**Темы, рассматриваемые в главе:**

- Человеческий фактор и тестирование
- Совершенствование процесса тестирования
- Что дальше

В главе 1 мы говорили о быстром тестировании как о структуре, построенной на основе следующих факторов:

- Исполнители
- Интегрированный процесс тестирования
- Статическое тестирование
- Динамическое тестирование.

До сих пор основное внимание уделялось второму строительному блоку структуры, а именно, процессу комплексных испытаний. Одна из причин повышенного внимания этому процессу связана с тем, что независимо от квалификации исполнителей, если они не имеют в своем распоряжении систематической, упорядоченной методики тестирования, то не смогут работать с максимальной отдачей. В первой части данной главы мы перенесем акцент на изучение человеческого фактора при тестировании. Следует отметить, что исследованию проблем, привносимым человеческим фактором в разработку программного обеспечения, посвящено немало книг, одной из которых является [41]. В своей книге *Peopleware (Кадровое обеспечение)* [14], которая стала классикой, Демарко (DeMarco) и Листер (Lister) рассматривают проблему человеческого фактора с точки зрения его влияния на разработку программных продуктов. В первой части данной главы будут обозначены наиболее важные аспекты этой темы, которые могут содействовать успеху тестовых работ, но в равной степени могут стать основной причиной их неудачи.

Несмотря на то что многие аспекты процесса отладки уже были обсуждены, придется затронуть еще один вопрос. В конце главы 5 отмечалось, что весь процесс тестирования нельзя кардинально перестроить. Совершенствование процесса тестиро-

вания должно быть планомерным процессом, разбитым на отдельные этапы. Вторая часть этой главы представляет собой обзор мероприятий, направленных на совершенствование процесса отладки.

Статическое и динамическое тестирование образуют третий и четвертый строительные блоки рациональной и эффективной методики тестирования. Эти две технологии тестирования основательно обсуждались во второй и третьей главах, а более подробное их исследование будет продолжено в главах 9 и 10.

# Человеческий фактор и тестирование

Барри Боем (Barry Boehm) в [21] утверждает, что личные качества исполнителей и отношения, установившиеся в коллективе, представляют собой резерв для повышения эффективности программного обеспечения. Другими словами, человеческий фактор оказывает гораздо большее влияние на эффективность программного обеспечения, чем любой другой отдельно взятый фактор. Боем доказывает это утверждение, применяя инструментальное средство COCOMO для оценки трудозатрат системного аналитика и программиста, при этом производительность каждого из них изменялась в 4 раза. Как показывает опыт, накопленный авторами, именно такой разброс производительности возможен во время выполнения тестовых работ, при разработке тестов и даже при прогоне тестов (т.е. при обнаружении дефектов). Если есть намерения выявить фактор, оказывающий максимальное влияние на способность вашего коллектива выполнять тестирование программных продуктов быстро и эффективно, в этом плане потребуется подвергнуть исследованиям в первую очередь деловые и личные качества членов группы тестирования.

В этом разделе мы покажем, какими качествами должен обладать тестировщик, дабы успешно справляться со своими обязанностями, Затем мы составим список ошибок, приводящих к снижению эффективности работ по тестированию, которые могут допускать тестировщики. Кроме того, будет рассмотрен ряд технологий опроса, ориентированных на специалистов по тестированию.

## Качества, которыми должен обладать специалист по тестированию, чтобы успешно справляться со своими обязанностями

Среди тестировщиков есть яркие представители, которые могут служить истинными примерами специалиста по тестированию программного обеспечения. Однако более привычным является вариант высококвалифицированной группы, которую составляют специалисты, обладающие различными навыками. В этом разделе будет составлен список качеств, которыми должен обладать идеальный тестировщик. Следует отдавать себе отчет, что в то время, как ни один человек не может быть носителем всех необходимых качеств, группа тестирования, будучи единым целым, может воплощать максимально возможное количество требуемых качеств. В основу этого списка положен опыт и наблюдения, но отнюдь не достоверные данные научных исследований; возможно, возникнет желание нарисовать собственный портрет идеального тестировщика, воспользовавшись предлагаемым списком в качестве отправной точки. На наш взгляд, идеальный тестировщик:

- Должен уметь разрушать программные продукты, не чувствуя при этом никаких угрызений совести. Поскольку тестирование выполняется с целью обнаружения дефектов, тестировщик не должен испытывать дискомфорта, обнаруживая ошибки в работе другого исполнителя.
- Должен уметь разрабатывать и выполнять пошаговые процедуры.
- Должен уметь описывать последовательность событий и конфигурацию системы, которые приводят к возникновению проблемы. Это включает способность четко документировать процедуры и результаты, умение устно передать информацию разработчикам, другим тестировщикам и руководству.
- Уметь критиковать и корректно воспринимать критику (например, умение так объяснить разработчикам суть дефектов, что с его слов их можно устранить).
- Обладать способностью приносить разработчикам и руководству плохие новости. Если в одиннадцать вечера выясняется, что не удается достичь готовности выпуска программного продукта, тестировщик должен быть готов сообщить руководству эту печальную новость.
- Уметь противостоять неослабевающему давлению (тестирование всегда является завершающей стадией любого процесса разработки и, как правило, протекает в стрессовых обстоятельствах).
- Обладать незаурядными умственными способностями, т.е. легко и быстро осваивать новые технологии.
- Быть терпеливым — быть готовым выполнять прогоны тестов столько раз, сколько нужно для того, чтобы снять проблему, после чего повторно выполнить тесты, чтобы убедиться в корректном устранении проблемы. (Между прочим, существенную помощь в этом случае оказывает именно автоматизация!)
- Обладать гибким мышлением — быть способным быстро переключиться на тестирование нового программного продукта или даже отказаться от испытания одного продукта в пользу другого, обладающего более высоким приоритетом.
- Обладать способностью одновременно видеть общую панораму и уметь при необходимости сосредоточиться на деталях; иметь широкий и динамичный кругозор.
- Быть экспертом в нескольких областях — группе тестирования могут потребоваться специалисты по базам данных, по коммуникациям, по сетевым технологиям, по тестированию GUI-интерфейсов, по инструментальным средствам тестирования, по сценариям автоматизации, а также специалисты из других областей.

Этот перечень достоинств высококвалифицированного тестировщика может с пользой применяться при приеме людей на работу и при оценке кандидатов на ту или иную должность. Если вы формируете коллектив для работы над новым проектом, рекомендуется подбирать людей таким образом, чтобы они соответствовали, по возможности, максимальному числу требований, фигурирующих в приведенном выше списке. Более подробную информацию о создании производственных коллективов можно найти в [14] и [33].

## Характерные ошибки

Если бы все группы тестирования проявляли качества, упомянутые в предыдущем разделе, проблем при проведении испытаний было бы значительно меньше. Однако лишь немногие коллективы тестировщиков в той или иной мере приближаются к идеалу, в связи с чем имеет смысл проанализировать наиболее распространенные ошибки, которые допускают тестировщики и которые ведут к снижению эффективности усилий, затрачиваемых на тестирование. Иногда достаточно простого предостережения в адрес партнеров по работе (или самому себе) о возможности таких ошибок и формирования стратегии совместного устранения последствий этих ошибок.

Вот список некоторых классических ошибок, допускаемых тестировщиками:

- **Предположение, что программа работает корректно.** Тестировщик всегда должен предполагать, что программа работает *некорректно.* Его обязанности заключаются в том, чтобы отыскать те моменты, которые программа выполняет *неправильно,* а не то, что она делает правильно. Любое отклонение от ожидаемого результата, вне зависимости от его значимости, должно рассматриваться как потенциальный признак дефекта. Независимо от того, насколько предупредительным и компетентным может быть программист, вы не должны отдавать свои симпатии этому программисту в ситуации, когда возникают подозрения о наличии дефекта.
- **Нежелание регистрировать каждую обнаруженную проблему.** Подобные ситуации часто возникают во время прогона продолжительного теста. Вы сталкиваетесь с некоторой незначительной проблемой, однако двигаетесь дальше, надеясь на то, что запомнили достаточно сведений, дабы зафиксировать их позже. Когда это "позже" наступает, вы либо забываете об этой проблеме, либо не можете воспроизвести действия, которые к ней приводят. Хороший способ не допускать такие ошибки состоит в ведении специального журнала записей, в котором фиксируются незначительные отклонения от нормы, даже те, которые на первый взгляд не имеют отношения к тестированию. Если вести точные записи, то в конце теста можно вернуться к обнаруженной проблеме и провести специальные исследования с целью определить, дефект ли это на самом деле.
- **Игнорирование или даже сокрытие проблемы.** Эта ошибка является частным случаем двух первых видов ошибок. Она чревата весьма пагубными последствиями. В этом случае известно, что программный продукт содержит дефект, однако почему-то принимается решение игнорировать его. Возможно, что в следующей сборке этого дефекта не будет. Возможно, что он не играет никакой роли. Возможно, по причине ограничений во времени проведение исследований дефекта невозможно. Такая ошибка может привести к тяжким последствиям, в том числе и вызвать просачивание дефекта в среду заказчика. Всегда следует помнить, что любой обнаруженный дефект может скрывать за собой целый ряд других, скрытых дефектов, если только последовательно не бороться с теми из них, которые находятся на виду.
- **Вы позволяете разработчикам уговорить себя не составлять сообщений о дефектах или игнорировать имеющиеся сообщения о дефектах, не имея на то достаточных оснований.** Нет ничего предосудительного в том, чтобы пого-

ворить с разработчиком и позволить ему убедить себя в том, что дефект присутствует в самом тесте, или же в том, что конфигурация тестовых средств не соответствует вероятной среде заказчика. Однако прежде чем принимать решение игнорировать тот или иной дефект, следует быть уверенным в правильности этого решения. В случае малейших сомнений относительно желания разработчика отказаться от регистрации дефекта или игнорировать то или иное сообщение о дефекте, нужно обсудить логику его рассуждений с коллегой-тестировщиком или с другим разработчиком.

- **Стремление не обострять отношения с разработчиком.** Этот вид ошибки имеет отношение к предыдущему типу, в то же время формы ее проявления могут быть другими. Если вы находите ошибку в чьей-то работе, это значит, что дефект засел в программном продукте, выпускаемом вашей компанией, и вся группа тестирования только выигрывает от того, что об этом ей стало известно, ибо чем раньше дефект будет устранен, тем лучше. Одним из достоинств эффективной системы отслеживания дефектов и еженедельных анализов дефектов является то, что они позволяют избегать конфликтов между людьми. Личные обиды, которые возникают, когда кто-то находит недостатки в чужой работе, можно сгладить, если профессионально применять соответствующие инструментальные средства и процедуры анализа дефектов.
- **Недостаточное внимание, которое уделяется планированию испытаний.** Если ваш процесс не поддерживается планированием испытаний, то элементарно может случиться так, что у вас останется слишком мало времени на подготовку к тестированию следующей сборки. Может также иметь место спешка при оценке затрат времени и ресурсов, необходимых для выполнения конкретной тестовой задачи. Планы, составляемые в спешке, обычно приводят к возникновению больших проблем при проведении испытаний.
- **Написание пространных отчетов о несуществующих проблемах.** Подобный вид бездарной траты времени находится на противоположном конце спектра по отношению к большинству рассмотренных выше ошибок. Вам спустили сверху квоту на отлов дефектов? Оценка вашего труда как-то связана с количеством обнаруженных дефектов? Если это так, то иногда трудно устоять перед искушением представить в качестве дефектов несуществующие проблемы. Поиск дефекта в программе, которая работает в соответствии с замыслом, требует больших непроизводительных затрата времени. Вы не только понапрасну тратите время на формулирование проблемы, вы отнимете время разработчиков и руководства проектом, которое они вынуждены будут отдать отслеживанию несуществующего дефекта и попыткам воспроизвести проблему. Тестировщики должны неукоснительно фиксировать каждую реальную проблему, с которой они сталкиваются, но ни а коем случае не поддаваться соблазну фиксировать те проблемы, которые, по их убеждению, будут проигнорированы. Если более 10% дефектов, обнаруженных конкретным тестировщиком, попадают в число несуществующих и будут игнорироваться, то этот тестировщик либо намеренно купился на эту уловку, либо ему явно не хватает знания деталей тестируемой программной системы.

## Как проводить опросы претендентов

Одним из способов формирования дееспособной группы тестирования является наем квалифицированных тестировщиков. И хотя подробное обсуждение стратегий и технологий опроса выходит за рамки данной книги, мы дадим краткую характеристику технологий опроса, ориентированных на специалистов по тестированию, которые, по мнению авторов данной книги, обеспечивают неплохие результаты. Все они представляют собой простые идеи, основанные на здравом смысле, однако если ранее ими пользоваться не доводилось, возможно, информация, полученная благодаря применению этих технологий, вызовет некоторое удивление.

- **Выясните, каким опытом работы в тестировании обладает кандидат.** Участвовал ли он когда-либо в составлении планов проведения испытаний? Если это так, то какая информация учитывалась в этом плане? Приходилось ли ему отображать тестовые случаи на технические требования? Пользовался ли они формальной системой отслеживания дефектов? Несколько вопросов с последующим их обсуждением позволяют достаточно быстро дать оценку опыта работы кандидата с процессами тестирования.
- **Если кандидат претендует на квалификацию в некоторой предметной области, спросите его, как он применял знания этой предметной области во время тестирования программного продукта.** Легко освоить технический сленг и рассуждать о предметных областях, однако толковое объяснение того, как кандидат разрабатывал сценарии автоматизации или конфигурировал локальную сеть или как использовал различные инструментальные средства, позволяют дать оценку его знаниям конкретных технологий.
- **Задайте вопросы, которые могут продемонстрировать, обладает ли кандидат перечисленными выше качествами, которые необходимы для того, чтобы успешно справиться со своими обязанностями.** Обычно вопросы, подобные "Достаточно ли вы умны для выполнения такой-то работы?" или "Достаточно ли вы терпеливы, чтобы выполнять обязанности, на которые претендуете?", едва ли дадут нужную информацию. Разумеется, можно предложить кандидату рассказать о предыдущей работе в других местах; такой рассказ может пролить свет на некоторые из этих качеств. Например, можно поинтересоваться: "Приходилось ли вам менять проектные приоритеты в тех случаях, когда требовалось немедленно перейти от тестирования продукта А к тестированию продукта В? Если да, то как это все происходило? Были ли у вас какие-либо затруднения при изучении технологии, связанной с продуктом В? С учетом приобретенного опыта, что у вас получилось хорошо? Что бы вы хотели изменить?" Если для формирующейся группы тестирования подыскивается работник, обладающий одним или несколькими из перечисленных выше качеств, стоит заблаговременно подготовить набор вопросов для выяснения области интересов кандидата.
- **Если вы хотите напять работника, отвечающего конкретным требованиям, можно предложить ему составить план действий в гипотетической ситуации и проанализировать предложенное им решение.** Например, если требуется исполнитель, умеющий составлять планы проведения испытаний и тестовые случаи для некоторого специального вида программного продукта, можно

разработать простой набор технических требований и спросить кандидата, какими видами тестов он воспользуется. Пребывая в поисках нужного решения вместе с кандидатом и делая при необходимости наброски на бумаге или устно обсуждая различные детали проблемы, следует получить представление о том, насколько хорошо кандидат справляется с задачей написания тестов.

- **Дайте оценку того, насколько кандидат склонен совершать перечисленные выше классические ошибки, создав ему соответствующие ситуации и проанализировав его ответы.** Например, можно поставить кандидата в ситуацию, когда на группу тестирования оказывается давление с целью сокращения сроков тестирования. Далее неплохо было бы задать ему приблизительно такие вопросы. Откажетесь ли вы от плана проведения испытаний в пользу проведения специализированного тестирования? Будете ли регистрировать проблемы, с которыми столкнетесь в процессе прогона тестов, или же только факты успешного или неудачного исхода? Удосужитесь ли вы потратить некоторое время на составление плана, указывающего, прогон каких тестов будет выполняться на завершающей сборке программного продукта, либо же сломя голову приступите к прогону тестов в надежде все завершить до того, как прозвенит завершающий звонок? Какие решения принимались в прошлом в аналогичных ситуациях?.. Можно предложить и другие ситуации, которые предполагают конфликт с разработчиками, или в которых требуется принять решение, если вдруг возникает подозрение, что коллега сообщает о несуществующих дефектах и проблемах.

В этом разделе затрагивается лишь несколько аспектов, имеющих отношение к кадровому обеспечению процесса тестирования. В то же время, несмотря на краткость изложения, нам удалось коснуться трех из пяти базовых принципов кадрового обеспечения, сформулированных Боемом (Boehm) [21]:

- Профессиональная пригодность
- Распределение работ в соответствии с квалификацией
- Продвижение по службе
- Пропорциональное распределение ресурсов внутри группы тестирования
- Постепенное сворачивание работ (с устранением несоответствий).

Принцип подбора кадров по профессиональной пригодности отражен в приведенном выше списке качеств идеального специалиста по тестированию. В этом списке также отражаются вопросы распределения работ в соответствии с квалификацией и пропорционального распределения ресурсов внутри группы тестирования. Не следует забывать о том, что ни один исполнитель не может обладать всеми профессиональными качествами, которые требуются от группы тестировщиков, в связи с чем кадровый состав группы тестирования должен быть сбалансирован так, чтобы покрывать все требования, предъявляемые к квалификации кадрового состава группы.

Принцип постепенного сворачивания работ предусматривает освобождение от обязанностей исполнителей, не способных внести положительный личный вклад в деятельность группы тестирования. Примером отрицательного вклада может служить ситуация, когда исполнитель своими действиями вносит такой беспорядок в

работу группы, что затрачиваемые на его устранение усилия приводят к существенному снижению производительности группы в целом. По-видимому, такие исполнители совершают некоторые ошибки, о которых речь шла выше. Скорее всего, они не обладают качествами, которые нужны специалисту по тестированию, дабы успешно справляться со своими обязанностями.

Принцип продвижения по служебной лестнице играет важную роль в тестировании программного обеспечения. Человек, которого принимают на работу в группу тестирования, должен знать, какая служебная карьера его ожидает. Все большее число компаний и специалистов рассматривают тестирование как вполне подходящую перспективу для служебного продвижения, в рамках которого отрываются возможности совершенствования в технических областях и приобретается опыт руководящей работы. Тестирование программного обеспечения — это удачное место для изучения производства программных продуктов на системном уровне. Поскольку оно затрагивает все аспекты продукции компании, оно может служить хорошей стартовой позицией для дальнейшей карьеры разработчика, а также специалиста по сопровождению продукта на площадках заказчика, по технической поддержке пользователей или по техническому маркетингу.

# Совершенствование процесса тестирования

Эффективный и отлаженный процесс тестирования нельзя получить в одночасье. Требуются месяцы, и даже годы планирования и напряженной работы, чтобы создать хорошо функционирующую организацию, способную выполнять работы по тестированию. При любой попытке усовершенствовать установившийся процесс тестирования с самого его основания нужно принять во внимание следующие моменты:

- Тестирование не может быть изолированным процессом. Тестирование программного обеспечения может быть эффективным только том случае, если оно интегрировано в жизненный цикл разработки программного обеспечения. Это означает, что попытки усовершенствовать сам процесс дадут мало пользы, поскольку совершенствование процесса тестирования должно выполняться в рамках более масштабных и всеобъемлющих работ.
- Эффективный процесс тестирования может быть создан только за счет его построения по стадиям. Существует несколько функциональных стадий или уровней любого процесса разработки. Эти уровни могут быть представлены моделями развития, к которым относятся: модель CMM (Capability Maturity Model — модель развития функциональных возможностей) программного обеспечения, разработанная институтом технологии программного обеспечения SEI (Software Engineering Institute), модель TMM (Test Maturity Model — модель развития тестовых возможностей), разработанная Иллинойским технологическим институтом, и модель TMM (Test Process Improvement — модель совершенствования процесса тестирования). Переход с одного уровня модели развития на другой должен происходить упорядочение Ни одна организация не может перейти с нижнего уровня развития сразу на верхний за счет адаптации всего лишь нескольких процедур или инструментальных средств; продвижение должно происходить постепенно, уровень за уровнем, и без каких-либо пропусков.

- Успешное совершенствование процесса затрагивает людей, которые с ним связаны. Организация процесса того или иного процесса — это далеко не академическое или техническое мероприятие; она требует от персонала внести определенные коррективы в свои рабочие обязанности и заставляет руководство пересмотреть свои взгляды на то, что можно ожидать от подчиненных. Для того чтобы любая инициатива, касающаяся улучшения процесса, оказалась успешной, в ее реализацию должны вносить посильный вклад все службы компании, от высшего руководства до исполнителей, занимающихся прогоном тестов в испытательной лаборатории. Как только процесс будет определен, персонал, которому надлежит его использовать, должен пройти специальную подготовку, дабы достичь необходимого уровня согласованности и производительности.

Первый пункт, утверждающий, что процесс тестирования не является независимым, очевиден на протяжении всего жизненного цикла разработки. В качестве' примера взаимной функциональной зависимости рассмотрим процесс выявления и формулирования технических требований. Как уже говорилось в предыдущих главах, тестирование должно начинаться с проверки формулировок требований. Формулирование требований представляет собой широкую межфункциональную деятельность, в которой принимают участие представители руководства высшего уровня, групп маркетинга, разработчиков, производителей и поддержки на стороне заказчика. Представители других групп, кроме группы маркетинга, могут не принимать участие в прямом обмене данными с заказчиком; в то же время совершенно ясно, что сформулированные требования оказывают влияние на работу всей группы разработки программного продукта.

Вторая особенность, связанная с тем, что эффективный процесс может строиться только постепенно, от стадии к стадии, рассматривается ниже, в контексте модели развития функциональных возможностей. Мы не будем исследовать все уровни и ключевые области процесса тестирования сквозь призму модели CMM, тем не менее, сосредоточимся на тех областях CMM, которые имеют отношение к принципам процессам тестирования, которые рассматриваются в данной книге. Упомянутые выше модели развития подробно рассматриваются в [19], [41], [38] и [29].

Третий пункт показывает, каким путем достигается совершенствование процесса тестирования. Исследованию этих вопросов посвящена завершающая часть данной главы.

## Модель развития функциональных возможностей программного обеспечения CMM

Институт технологий программного обеспечения SEI (Software Engineering Institute) — это проектно-исследовательская организация, основанная в 1984 году Министерством обороны США и финансируемая федеральным правительством США. Она поддерживается университетом Карнеги-Меллон. Информацию, касающуюся института SEI и модели CMM, можно получить на Web-сайте института SEI по адресу: *www. sei.cmu. edu.*

Модель развития функциональных возможностей является основой, которая может использоваться для получения оценки, в какой степени заданная организация

способна изготовлять программное обеспечение. Она классифицирует процесс разработки программного обеспечения по пяти уровням развития: начальный, повторяемый, определяемый, управляемый и оптимизирующий. Краткие характеристики каждого из этих уровней приводятся в таблице 6.1.

Характеристики, перечисленные в таблице 6.1, описывают ключевые свойства уровней 1-5. Уровень детализации, предложенный в таблице 6.1, намеренно выбран низким. Подобный выбор обусловлен необходимостью дать общее представление об организациях, находящихся на каждом из пяти уровней. Например, организация, расположенная на уровне 1 модели CMM, т.е. на начальном уровне, разрабатывает программное обеспечение с использованием неопределенных и непланируемых процессов. Скорее всего, эта организация не способна соблюдать сроки разработки, обнаруживать и устранять все дефекты программного продукта перед его выпуском и укладываться в рамки сметной стоимости. Организации, достигшие уровня 2 модели CMM, вложили существенные средства в совершенствование реализуемых ими процессов, и во время планирования, оценки и составления календарных планов руководствуются четкими принципами управления ходом проектирования, тем самым достигая поставленных целей. Можно вполне рассчитывать на то, что организации второго уровня, по сравнению с организациями первого уровня, лучше справляются с задачами соблюдения срока разработки, обнаружения и исправления дефектов, равно как и более точно управляют окончательной стоимостью проекта.

По сравнению с хаосом, присутствующим на уровне 1, уровень 3 привносит существенные улучшения. На уровне 3, т.е. на определяемом уровне, тестирующая организация, скорее всего, реализует большую часть методов, обсуждаемых в данной книге, а именно: управление требованиями, достоверные оценки, составление графиков работ, управление конфигурациями, а также инспекции и пересмотры. Согласно модели CMM, определяемый уровень представляет собой высокий уровень эффективности организации, однако управление процессом осуществляется скорее на основе качественных, а не количественных оценок. На этом уровне все еще не установлены надежные системы измерений, пригодные для оценки качества программного продукта или эффективности процесса.

Два последних уровня модели CMM предусматривают плановые и текущие измерения качества продукта и эффективности процесса. Основные различия между уровнями 4 и 5 заключаются в том, что основное внимание на уровне 5 уделяется не качеству программного продукта, а эффективности процессов. Продвижение к уровню 5 означает, что организация собрала достаточное количество данных, позволяющих ей воспользоваться средствами настройки своих процессов на достижение оптимальной эффективности. Одним из ключевых свойств уровней 4 и 5 является применение метрик тестирования. Использование упомянутых метрик подробно обсуждается в главе 11.

В дополнение к описанию пяти уровней функционального развития организаций, занимающихся разработкой программного обеспечения, модель CMM определяет 18 областей обработки, распределенных по всем пяти уровням. Области KPA (Key Process Areas— ключевые области обработки) для заданного уровня функционального развития суть полностью действующие процессы, которые в любой момент времени должны находиться в полной функциональной готовности. Они направлены на то, чтобы организация могла претендовать на соответствующий уровень.

**Таблица 6.1. Характеристики модели развития функциональных возможностей**

| Уровень развития | Наименование уровня | Характеристики |
|---|---|---|
| Уровень 1 | Начальный | В большинстве случаев процесс не определен и не рассчитан на серийное производство. Как правило, отсутствуют формализованные процедуры, планы проектов, а сметы не составляются. Если все же такие процедуры существуют, то отсутствует такой механизм управления, который обеспечил бы их использование. В критических ситуациях формальные процедуры не используются. Контроль внесения изменений практически отсутствует. Кодирование и тестирование выполняются без определенного плана. Проекты, пересмотры и анализ результатов тестирования считаются непозволительной роскошью. Руководство организации имеет смутное представление о проблемах. |
| Уровень 2 | Повторяемый | Разработка ведется в соответствии с планом. Установлена система управления проектом, обеспечивающая поддержку планирования и отслеживания. Организована группа обеспечения качества, дающая гарантию руководству, что работа ведется в рамках заданного процесса и в соответствии с планом. Для программного обеспечения проводится контроль внесения изменений. Сметные требования и требования соблюдения графиков работ поддерживаются до тех пор, пока не внедряются новые инструментальные средства и новые методы, и если не предпринимаются крупные изменения внутри самой организации. |
| Уровень 3 | Определяемый | В процессы, реализуемые на уровне 2, вносятся дополнительные усовершенствования. Организуется группа поддержки процесса, которая сопровождает совершенствование текущего процесса. Определен жизненный цикл разработки, который приводится к конкретным нуждам организации и к текущему проекту. Постоянно применяется семейство методов проектирования, в числе которых методы формального проектирования и методы тестирования. Организация, поднявшаяся до этого уровня, приобретает фундамент, обеспечивающий серьезное и долгосрочное развитие. Однако каким бы ни был эффективным этот уровень, он остается качественным уровнем. На этом уровне существует мало средств для количественной оценки того, что делается, и насколько эффективным оказывается процесс. |
| Уровень 4 | Управляемый | В дополнение к усовершенствованиям, внесенным на предыдущем уровне, реализуется целый ряд измерений с целью предоставления возможности выдать количественную оценку стоимости и достоинств главного процесса. Устанавливается и поддерживается база данных процесса. Отслеживается продвижение в направлении достижения целей качества для конкретных программных продуктов, причем соответствующие сведения направляются руководству. Основное внимание на этом, впрочем, как и на всех предыдущих уровнях, уделяется качеству продукта. |

***Окончание табл. 6.1***

| Уровень **развития** | **Наименование уровня** | **Характеристики** |
|---|---|---|
| Уровень 5 | Оптимизирующий | На этом уровне происходит смещение акцентов. Основное внимание теперь уделяется не качеству программного продукта, но оптимизации процесса. На этом уровне становятся доступными данные, которые можно использовать для настройки самого процесса, т.е. можно не принимать во внимание ограничения, накладываемые одним или двумя какими-либо проектами. На оптимизирующем уровне организация получает в свое распоряжение средства, позволяющие выявлять и исправлять наиболее слабые элементы процесса. |

Например, названиями областей KPA для уровня 2 [38] являются:

- Управление требованиями
- Планирование проекта программного обеспечения
- Контроль и отслеживание разработки проекта программного обеспечения
- Управление договорами с субподрядчиками на разработку программного обеспечения
- Обеспечение качества программного обеспечения
- Управление конфигурацией программного обеспечения.

Подробное описание областей KPA мы оставляем на публикации, посвященные моделям CMM. В то же время может оказаться полезным концептуальное отображение областей KPA уровня 2 на процессы тестирования, описание которых было дано в предыдущих главах настоящей книги. Попытка такого отображения предпринимается в следующем разделе.

## Как модель CMM соотносится с быстрым тестированием

Путь от уровня 1 к уровню 2 в соответствии с моделью CMM лежит через реализацию областей KPA, список которых можно найти в предыдущем разделе. Первым рассматривается процесс *управления требованиями.* Роль, которую играет группа тестирования в управлении техническими требованиями, была главной темой главы 2. В этой главе отмечалось, что группа тестирования должна получить доступ к точным спецификациям требований на ранних этапах разработки программного продукта, чтобы иметь основу для составления плана проведения испытаний и для проектирования тестов. Были приведены аргументы в пользу участия группы тестирования в процессе формулирования технических требований за счет выполнения статического тестирования требований на предмет полноты, непротиворечивости, осуществимости и контролепригодности. Кроме того, была также обоснована необходимость использования матрицы прослеживаемости требований с целью отображения каждого требования на тесты, отдельные компоненты проекта и на программные коды.

В главе 8 приводится описание процесса JAR, представляющего собой методологию выявления требований, которая в полной мере интегрирует статическое тестирование и процесс формулирования требований. Интегрирование статического тестирования с определением и прослеживанием требований на этапах планирования

испытаний и разработки тестовых случаев — это базовые принципы, которыми следует руководствоваться при определении роли группы тестирования в управлении требованиями.

Вторая область KPA, согласно модели CMM принадлежащая уровню 2, — это *планирование проекта*. Применительно к организациям, занимающимся тестированием, эта область охватывает оценку трудозатрат, разработку графиков работ и планов проведения испытаний. Составление плана проведения испытаний и оценка трудозатрат достаточно подробно рассматривались в главе 3, а вот дополнительная информация по технологиям оценки трудозатрат содержится в главе 12. Пример плана проведения испытаний представлен в главе 14. В результате обсуждения вопросов, связанных с планированием испытаний, был сделан вывод, что к основным видам деятельности, которые составляют процесс планирования испытаний, относятся:

- Определение стратегии тестирования
- Определение состава испытательного комплекса (программные и аппаратные средства)
- Оценка трудозатрат (трудовые ресурсы и графики работ)
- Оценка рисков срыва графиков работ и подготовка плана смягчение последствий от овеществления рисков
- Подготовка и пересмотр документов, содержащих план проведения испытаний.

В то время как очередность, в которой организация рассматривает области KPA, зависит от локальных условий, есть все основания сначала решать вопросы управления требованиями, а после этого переходить к планированию проекта. Причина очень проста — в основе этого плана лежат требования. По мере изменения требований, план должен модифицироваться, дабы соответствовать произведенным изменениям.

Третьей областью KPA в модели CMM является *контроль и отслеживание разработки проекта программного обеспечения*. Для группы тестирования это означает, что информация о ходе тестирования и ожидаемом качестве продукта в том виде, в каком она представлена в сообщениях о дефектах, должна в любой момент быть доступной руководству. Эти вопросы рассматривались в главе 5 при обсуждении отчета о ходе работ по тестированию и отчета об обнаруженных дефектах. Если быть точным, то новейшая информация доводится до сведения заинтересованных лиц на еженедельных (а в критических ситуациях, ежедневных) совещаниях. Если эта информация пересылается на внутренний Web-сайт, то действия группы тестирования должны соответствовать предназначению этой области KPA. Поскольку для отслеживания проекта должен существовать план, вполне очевидно, что эта область KPA зависит от корректного выполнения KPA, предусмотренного планированием проекта.

Область KPA, именуемая *управлением договорами с субподрядчиками на разработку программного обеспечения,* до сих пор еще не рассматривалась. Тем не менее, она может иметь очень важное значение для групп тестирования, которые используют услуги субподрядчиков во время выполнения различных видов тестовых работ: при составлении планов проведения испытаний, при разработке тестов и/или при прогоне тестов. Нередки случаи, когда одна или большее число географически удаленных групп заключают договор на выполнение определенной части работ по тестирова-

нию. Необходимо отметить три соображения относительно привлечения субподрядчиков. Во-первых, они могут привлекаться к сотрудничеству с целью оценки трудозатрат и планирования испытаний на начальной стадии проекта. Желательно, чтобы они принимали активное участие в работах по оценке и планированию. Во-вторых, все ожидаемые результаты планов должны быть включены в договор, который согласован и утвержден всеми заинтересованными сторонами. Наконец, задача контроля и отслеживания разработки проекта должна включать работу субподрядчиков.

Пятой областью KPA, которая фигурирует в списке для уровня 2 модели CMM, является *обеспечение качества.* Основной функцией поддержки качества на уровне 2 является текущий контроль разработки программного продукта с тем, чтобы обеспечить строгое соблюдение требований процесса и получить оценку качества программного продукта. Группа обеспечения качества служит для руководства своего рода окном, позволяющим получить представление о том, как создается проект. Группа обеспечения качества не должна зависеть от разработки и управления в том смысле, что она должна обеспечить независимый, объективный взгляд на положение вещей. Под "независимым" понимается тот факт, что группа обеспечения качества предоставляет отчеты руководству отдельно от группы разработки. И хотя группу тестирования могут попросить выступать в роли группы обеспечения качества, в рамках данной книги подобная роль не предполагается ни в одном из процессов. Все виды тестовой деятельности, такие как формальные инспекции и неформальные перепроверки, в данном случае рассматриваются как составные части статического тестирования.

Последней областью KPA в списке для уровня 2 модели CMM является *управление конфигурацией.* Основное назначение управления конфигурацией состоит в размещении избранных рабочих продуктов в безопасном хранилище, которое контролируется механизмом управления версиями. Естественно, под управление конфигурациями должны подпадать документы с техническими требованиями, проектные документы и программные коды. В перспективе тестирования план проведения испытаний, тестовые случаи и результаты тестирования (включая сообщения о дефектах) так же передаются под управление конфигурациями. Мы обсуждали это в главах 3, 4 и 5, равно как и напоминали о том, что инструментальное средство управления тестированием является хорошим вкладом для решения конкретных задач управления конфигурацией силами группы тестирования.

## Возможности совершенствования процессов

Независимо от того, используется ли в качестве базы совершенствования процессов модель CMM, или принят какой-то другой подход, для решения задачи совершенствования процессов обычно применяется систематизированный и упорядоченный подход, который предусматривает выполнение следующих действий:

- **Оценка состояния текущего процесса.** Первое, что следует предпринять, так это определить, где вы находитесь по отношению к принятой базовой структуре. Для получения оценки можно привлечь консультантов извне, которые прошли специальную подготовку и способны дать оценку организации в контексте модели развития функциональных возможностей, подобной CMM. С другой стороны, внутреннюю оценку можно провести в форме послепроектного обзора, который часто предназначен не только для праздничного подведе-

ния итогов окончания проекта, но и для того, чтобы определить, что сделано хорошо, а что должно совершенствоваться. Обе оценки, внутренняя и внешняя, требуют полной поддержки со стороны верхнего эшелона руководства организации и участия специалистов, которым надлежит внедрить предлагаемые изменения и пользоваться ими в дальнейшем.

- **Определение областей, требующих совершенствования.** По результатам оценки текущего процесса должен быть составлен список его областей, в которых необходимо провести усовершенствования. При этом важно определить их приоритеты и зафиксировать этот порядок в соответствующих планах. Попытки внести изменения сразу во все области этого списка часто приводят к глубоким разочарованиям.
- **Планирование работ по совершенствованию процессов.** Необходимо привлечь к планированию специалистов, занимающихся внедрением усовершенствований, что позволит учесть их предложения и заручиться их поддержкой. Такой план должен предусматривать внедрение средств измерения степени продвижения по направлению к целям совершенствования процесса. В контексте модели развития функциональных возможностей, эта мера продвижения к цели по прошествии 12 или 18 месяцев может принять форму другой формальной оценки, с промежуточными контрольными точками, позволяющими отслеживать движение вперед.
- **Внедрение изменений и мониторинг их эффективности.** После создания плана, внедрения всех его пунктов, получения отдачи и выяснения корректив следует всячески сохранять движение вперед, установив его в качестве основного курса развития.

Для совершенствования процесса необходимо время. На переход с уровня 1 на уровень 2 у крупных организаций уходит от двух до трех лет. Продвижение небольших организаций может оказаться более быстрым (анализ совершенствования небольших проектов и организаций можно найти в [38]).

# Что дальше

Данная глава завершает первую часть, посвященную подходу к тестированию программного обеспечения, на который мы ссылаемся как на быстрое тестирование. Быстрое тестирование есть структура, построенная на основе:

- Персонала
- Интегрированного процесса тестирования
- Статического тестирования
- Динамического тестирования.

В этой главе мы обсуждали кадровый аспект тестирования и проблемы совершенствования самого процесса тестирования. Касаемо персонала был представлен список качеств, которыми должен обладать специалист по тестированию, перечислены ошибки, которые часто допускают тестировщики, и даны советы по интервьюированию потенциальных работников. Все это является предпосылками создания сбалансированной группы тестирования.

Выше отмечалось, что какой бы высокой квалификацией ни обладали специалисты, если в их распоряжении не будет систематизированного и упорядоченного процесса тестирования, они не смогут работать с максимальной отдачей. Мы сформулировали три базовых принципа, которые следует иметь в виду, реализуя свои стремления усовершенствовать процесс тестирования:

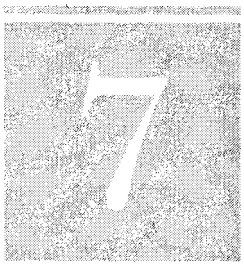
- Тестирование не может быть независимым процессом.
- Эффективный процесс тестирования может быть построен только постепенно, стадия за стадией.
- Чтобы совершенствование процесса привело к успеху, необходимо учитывать мнение конечных исполнителей.

В качестве примера пошагового подхода к совершенствованию процесса приводилась модель CMM программного обеспечения, разработанная институтом SEI. При этом некоторые ключевые области обработки были отображены на тестирование.

В следующей части книги предлагаются хорошо зарекомендовавшие себя на практике советы, технологии и примеры, которыми можно воспользоваться для повышения эффективности тестирования. Однако, как мы убедились на базе материала этой главы, процесс разработки программного обеспечения не поддается быстрым изменениям. Практически воплощая идеи, изложенные в первых двух частях книги, наверняка возникнет желание воспользоваться плановым пошаговым подходом к совершенствованию. При этом изменения в первую очередь будут вноситься в те части процесса, которые обеспечат наибольший эффект. Примеры организации ключевых тестовых работ можно найти в третьей части книги.


# Технологии быстрого тестирования и советы

# Введение в технологии тестирования и советы



Темы, рассматриваемые в главе:

- Область применения технологий тестирования
- Жизненный цикл разработки
- Преимущества быстрого тестирования
- Определение статического тестирования
- Определение динамического тестирования
- Жизненный цикл дефекта
- Формальные этапы тестирования
- Обязанности членов команды тестирования
- Что дальше

## Область применения технологий тестирования

Для более наглядного представления области применения технологий тестирования в таблице 7.1 приводится список технологий, которые используются при разработке тестовых задач. Некоторые из них являются статическими, другие — динамическими технологиями тестирования. Но можно ли с первого взгляда определить, к какой категории относится та или иная технология? Хотя этот список явно неполон, он призван убедить, что при использовании только нескольких из этих технологий в проекте могут быть допущены программные ошибки, которые легко могли бы быть обнаружены в случае применения одной из других технологий.

В следующих пяти главах приведены описания этих терминов, цели применения технологий и примеры использования части из них в проектах. Чтобы читателям было легче решить, когда следует применять те или иные технологии, в некоторых главах рассматриваются примеры, взятые из реальной жизни или построенные на основе опыта авторов.

**Таблица 7.1 Технологии тестирования**

| | |
|---|---|
| Аппаратное тестирование | Тестирование ветвей |
| Нагрузочные испытания | Тестирование сегментов |
| Испытания в утяжеленном режиме | Тестирование функциональных возможностей |
| Испытания на долговечность при хранении | Тестирование модулей |
| Тестирование конфигурации | Испытания баз данных |
| Тестирование совместимости | Логическое тестирование |
| Тестирование возможностей установки | Аттестационные испытания |
| Испытания надежности | Проверочные испытания (верификация) |
| Испытания на пригодность к обслуживанию | Контрольные (проверочные) испытания |
| Тестирование документации | Сбор претензий и пожеланий |
| Процедурное тестирование | Контроль синхронизации |
| Приемочные испытания | Тестирование обращений к подпрограммам |
| Климатические испытания | Испытания работы в многозадачном режиме |
| Испытания в режиме "черного ящика" | Испытания последовательности событий |
| Испытания в режиме "белого ящика" | Имитационная проверка |
| Входной контроль | Эксплуатационные испытания |
| Испытания на соответствие стандартам | Проверка точности |
| Сравнительные испытания | Испытания устойчивости |
| Испытания граничных значений | Тестирование на непротиворечивость |
| Испытания на наличие побочных эффектов | Проверка ранее внесенных изменений |

# Жизненный цикл разработки

Диаграмма шарнирно-каскадной модели, представленная на рис. 7.1, помогает понять, когда следует применять статическое тестирование, а когда — динамическое тестирование. Эта диаграмма шарнирно-каскадной модели не противоречит приведенной в конце главы 1, хотя рис. 7.1 и содержит несколько дополнительных аббревиатур, которые представляют основные термины, используемые в этой и последующих главах. Документация, служащая основой для проведения комплексных, системных и приемосдаточных испытаний, создается, соответственно, на этапах рабочего проектирования, эскизного проектирования и разработки технического задания. Данная взаимозависимость показана двунаправленными стрелками, и именно на этих этапах некоторые проекты испытаний начинают давать сбои. Если код не удовлетворяет требованиям технического задания, причиной программных ошибок могут послужить следующие три обстоятельства:

- Технические требования были изменены, но это не было отражено в документации
- Один из случаев тестирования, сценариев тестирования или результатов испытаний содержал ошибку, которая привела к ложному обнаружению отказа
- Код содержит программную ошибку

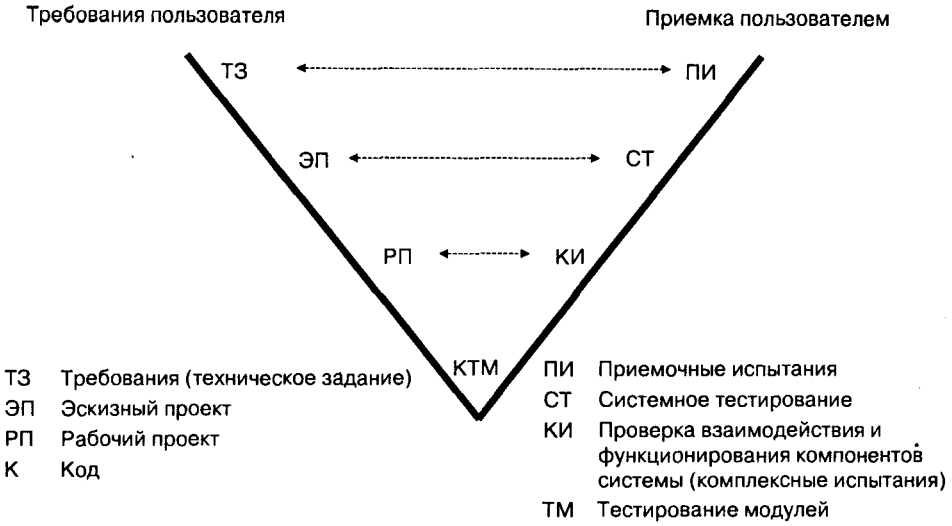



*Рис. 7.1. Диаграмма шарнирно-каскадной модели.*

ТЗ Требования (техническое задание)
ЭП Эскизный проект
РП Рабочий проект
ПИ Приемочные испытания
СТ Системное тестирование
КИ Проверка взаимодействия и функционирования компонентов системы (комплексные испытания)
ТМ Тестирование модулей

*Рис. 7.1. Диаграмма шарнирно-каскадной модели.*

Обратите внимание, что в двух первых случаях ошибка должна обнаруживаться средствами статического тестирования. Эти ошибки возникают вследствие несоблюдения жизненного цикла разработки. Персонал, выполняющий быстрое тестирование на ранних этапах разработки программного обеспечения (от разработки технического задания до написания кода), обеспечит наилучшую гарантию предотвращения подобных неудач в процессе тестирования.

В качестве общего правила можно утверждать следующее: технологии динамического тестирования должны применяться разработчиками программного обеспечения, начиная с проверки вновь созданного кода на этапе тестирования кода и модулей (ТКМ) или раньше, если предполагалось создание прототипа. Технологии статического тестирования применяются как на этапе разработки, так и на этапе тестирования.

Во время разработки некоторых проектов тестировщики не приступают к работе до тех пор, пока не будет завершен этап тестирования кода и модулей (ТКМ) и пока полностью не будет написан исходный код. Фактически, иногда тестировщики начинают свою деятельность лишь после частичного завершения этапа комплексных испытаний (КИ). Если в вашей организации используется именно такой поход, значит, в ней не исповедуется философия быстрого тестирования. Если специалисты по тестированию не участвуют в разработке планов испытаний, случаев тестирования, сценариев тестирования, не занимаются анализом результатов тестирования и статическим тестированием документации до и во время этапа ТКМ, стоит задуматься о напрасных потерях времени разработки, которые вызваны необходимостью устранения ошибок, допущенных на ранних этапах жизненного цикла разработки программного обеспечения. Кое-кто полагает, что источником всех ошибок является исходный код. Это совершенно неверно. Например, ошибки, допущенные на этапе разработки технического задания (ТЗ), могут вылиться в месяцы напряженной работы над архитектурными интерфейсами, низкоуровневого проектирования, создания кода и на-

писания документации. И лишь потом станет ясно, что полученный результат — вовсе "не то, что нам требовалось" или что "это выполняется не так, как нам требовалось". Мы часто упрекаем команду разработки технического задания, но кто из ее персонала способен обнаружить подобные ошибки и избежать их на этапе разработки ТЗ? Приходится лишь стыдиться, что никто из тестировщиков не принимал участия в работах этого этапа и, соответственно, попросту не мог обнаружить и устранить такие ошибки простым зачеркиванием или отправкой листа бумаги в корзину. Сколько денежных средств расходуется напрасно только потому, что тестировщики не участвуют в ранних этапах жизненного цикла разработки? В главе 8 рассматривается технология статического тестирования, получившая название совместной разработки требований к приложению (joint application requirements, JAR), которая служит примером следования философии быстрого тестирования, т.е. того, как необходимо сочетать выработку технических требований с их тщательным тестированием.

На ранних этапах жизненного цикла для описания того, что предположительно должно выполнять программное обеспечение (т.е. требования), используются слова обычного языка. Эти словесные описания превращаются в представленные в той или иной графической форме архитектурные диаграммы. Подсистемы и подблоки определяются соответствующими описательными фрагментами ТЗ. В ходе дальнейшего жизненного цикла разработки потоки операций и данных описываются другими графическими формами. Это же касается и таблиц, описывающих имена и отношения, которые будут положены в основу баз данных и запросов. После того, как логика программы полностью определена, для проверки полноты отражения программистами всех возможных ситуаций часто используются диаграммы состояний. Ошибки вносятся во время каждого такого преобразования из одного языка описания задачи в другой. Задачей тестировщиков является обнаружение ошибок подобного рода.

Стратегии разработки тестовых случаев естественным образом делятся на две категории, в зависимости от того, с чем тестировщику приходится иметь дело:

1. **Тестирование "черного ящика".** Если известны конкретные функции, которые должен выполнять данный продукт, можно прогнать тесты, подтверждающие полную работоспособность каждой из функций. Термин "черный ящик" просто означает, что при разработке тестовых случаев тестировщики ничего не знают о внутренней структуре или коде. Применяемые во время тестирования "черного ящика" технологии обычно называют технологиями динамического тестирования, и многие из них рассматриваются в главе 10.

2. **Тестирование "белого ящика".** Если известны особенности внутренней работы продукта, можно выполнить тесты, подтверждающие, что внутренняя работа продукта происходит в соответствии со спецификацией, а все внутренние компоненты используются правильно. Термин "белый ящик" означает, что при разработке тестовых случаев тестировщики используют любые доступные сведения о внутренней структуре или коде. Технологии, применяемые во время тестирования "белого ящика", обычно называют технологиями статического тестирования, и многие из них исследуются в главе 9.

На поздних этапах жизненного цикла программного обеспечения, которые представлены правой половиной шарнирно-каскадной модели, скомпилированный программный код используется для управления компьютерной системой, периферийны-

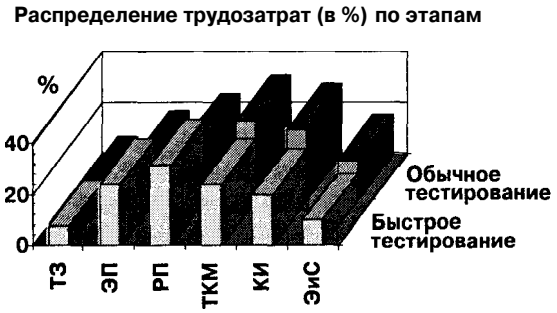
ми устройствами и подсистемами обмена данными с целью реализации всех функций, которые задокументированы как требования. При этом должны удовлетворяться все архитектурные и проектные спецификации. Часто именно на этом этапе приложение запускается первый раз и может быть подвергнуто испытаниям по производительности, совместимости с несколькими платформами, применимости, пригодности к установке и другим формам и технологиям динамического тестирования из рассмотренных в главе 10. Теперь все подсистемы и подблоки, описанные в архитектурной и проектной документации, реализованы и скрыты внутри компьютерного кода.

# Преимущества быстрого тестирования

Многие технологии быстрого тестирования могут применяться на ранних этапах жизненного цикла разработки программного обеспечения, как только становятся известны технические требования. В то же время, конкретные варианты этих технологий статического тестирования часто повторяются на более поздних этапах жизненного цикла (глава 8 посвящена технологии быстрого тестирования на этапе определения технических требований). Естественно, возникает вопрос: "Не связано ли привлечение специалистов по тестированию на ранних этапах жизненного цикла разработки с увеличением затрат?". На рис. 7.2 показана модель оценки стоимости разработки программного обеспечения (исследованиям этой модели оценки стоимости программного обеспечения посвящается глава 12), которая наиболее точно согласуется с экспериментальными данными по соотношению затрат для случаев обычного и быстрого тестирования. Сумма значений трудозатрат (в процентах), приведенных в таблице слева от этого рисунка, превышает 100%. Сумма трудозатрат этапов разработки, а именно — ЭП, РП, ТКМ и КИ, составляет 100%, и на диаграмме эти суммарные трудозатраты представлены областью, расположенной ниже кривой "разработка — трудозатраты". Традиционно при этом трудозатраты на разработку ТЗ не учитываются, поскольку разработка ТЗ должна быть завершена до того, как можно будет планировать трудозатраты этапов разработки. Значение трудозатрат, планируемых для этапа разработки ТЗ, составляет 8% для быстрого тестирования против 7% для обычного тестирования, причем это значение рассчитывается по отношению к трудозатратам, представленным областью под кривой "разработка — трудозатраты". В структуре трудозатрат на разработку программного обеспечения традиционно принято также не учитывать трудозатраты в течение первого года после поступления продукта на рынок (иногда этот этап называют этапом эксплуатации и сопровождения (ЭиС)). Планируемые трудозатраты этого этапа составляют 10% для быстрого тестирования и 15% для обычного тестирования (в процентах от общей суммы трудозатрат на разработку).

Обратите внимание, что хотя на диаграмме этапы разработки для случаев быстрого и обычного тестирования совпадают, из секторной диаграммы видно, что общая сумма трудозатрат при использовании технологии быстрого тестирования составляет 56,25% от общей суммы трудозатрат в случае применения обычного тестирования (36%/64%). Поскольку трудозатраты на разработку проектов с применением быстрого тестирования меньше трудозатрат на разработку проектов с применением обычного тестирования, время поставки программных продуктов на рынок также существенно сокращается.

| | Быстрое тестирование | Обычное тестирование |
|---|---|---|
| ТЗ | 8 | 7 |
| ЭП | 24 | 17 |
| РП | 32 | 24 |
| ТКМ | 24 | 31 |
| КИ | 20 | 28 |
| ЭиС | 10 | 15 |




Преимущества быстрого тестирования

- Стоимость
- Оптимальный календарный план
- Качество продукта
- Простота управления процессом разработки
- Учет интересов клиентов
- Чувство уверенности у разработчиков
- Повторное использование продукта

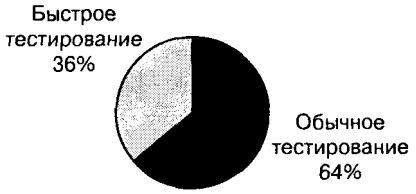



*Рис. 7.2. Сравнение трудозатрат* ***при*** *быстром и обычном тестировании.*

Представьте себя в качестве руководителя разработки проекта с использованием технологии быстрого тестирования. Вы ведете беседу с руководителем проекта, в котором применяется обычное тестирование. Вы: "Время разработки и объем трудозатрат моего нового проекта примерно в два раза меньше времени и трудозатрат, требуемых на создание программного продукта примерно такого же объема." Руководитель другого проекта: "Ха, готов поспорить, что качество вашего продукта никуда не годно." Вы: "На самом деле, как это ни странно, в программе оказалось всего около одной трети скрытых ошибок, из числа допущенных в предыдущем проекте аналогичного объема." Руководитель другого проекта: "Во имя всех святых, но как вы этого добиваетесь?" Вы: "Мы применяем технологию быстрого тестирования с первого до последнего дня разработки, параллельно и в комплексе с самой разработкой. И это действительно окупается. Мы перехватываем контракты у конкурентов, и наши конечные пользователи также остаются в выигрыше!"

# Определение статического тестирования

**При** статическом тестировании для визуальной экспертизы или автоматической обработки документации/кода программы с целью обнаружения ошибок используются только текстуальные и/или графические формы программного обеспечения. Текстовые или графические процессоры наподобие компиляторов, программ проверки перекрестных ссылок, эмуляторов дискретных событий, программ вывода на печать, программ статического контроля, программ распечатки ветвей и счетчиков цикломатической сложности — все они относятся к одной категории и носят название статических процессоров. Многие из них представляют собой исключительно важные инструментальные средства тестировщика программного обеспечения. Существует

ряд выполняемых вручную процессов, применяемых при таких видах статического тестирования, как экспертизы, контрольные списки и оценка проектов. Этим вопросам посвящена глава 8. Ошибки желательно обнаруживать до или в момент их внесения в жизненный цикл разработки программного обеспечения. Хорошим примером своевременного выявления/исправления ошибок может служить программа динамической проверки орфографии, выполняющаяся внутри текстового процессора. Эта программа активно ищет текстовые строки, отсутствующие в текущем словаре, и после обнаружения такой строки, заменяет ее наиболее близким по написанию словом или выделяет ее, чтобы пользователь смог выбрать правильное слово из списка слов с аналогичным написанием. Хотя статическое тестирование может применяться в течение всего жизненного цикла разработки и сопровождения, оно особенно полезно на этапах, которые предшествуют созданию выполняемого кода.

## Определение динамического тестирования

Динамическое тестирование суть обнаружение ошибок с помощью компьютера или эмулятора компьютера, предназначенного для выполнения программы, которая была создана разработчиками в соответствии с техническими требованиями. Динамическое тестирование соответствует естественному стремлению, возникающему при наличии выполняющей программы. А именно — предоставить компьютеру возможность помочь в обнаружении ошибок путем выполнения ряда примеров или сценариев, организованных в форме тестовых случаев. Ориентированные на пользователя экраны, которые, быть может, высмеивались на этапах разработки ТЗ и проектирования, неожиданно начинают оказывать влияние на характеристики производительности, которые, возможно, были неясны во время проводимого ранее анализа архитектуры. В процессе выполнения динамических тестов тестировщик действительно может поставить себя на место пользователя и выполнить тест так, как любой пользователь будет использовать программу. Сначала специалисты по тестированию могут обратить внимание на временную синхронизацию и точность результатов выполнения программы. В случае разработки новой подсистемы, прежде всего, активизируются другие подсистемы, обеспечивающие доступ к базе данных совместного использования, обмен данными между подсистемами или выполнение любых других функций, добавленных новой подсистемой.

Тестирование на совместимость предполагает выполнение одних и тех же тестов для программы на нескольких компьютерных платформах и/или на нескольких конфигурациях. Для сравнения, выполнение регрессивных тестов предполагает прогон одних и тех же тестов по отношению к (я+1)-ой версии программы и сравнение полученных результатов с результатами тестирования №ной версии. Вводя различные входные данные и наблюдая за изменением выходных данных, поведением и изменением производительности компьютерной системы, тестировщики могут решить, готова ли программа к началу использования конечными потребителями. Не всегда это столь просто, как кажется, поскольку необходимо учитывать очень много переменных, таких, например, как правильность работы оборудования, корректность функционирования компиляторов, кода динамической поддержки и операционной системы, под управлением которой выполняется приложение, пригодность данных для приложения и множество других факторов. Каждый из них может служить одной из

основных причин возникновения вероятных ошибок, которые должны быть обнаружены во время динамического тестирования.

Некоторые приложения используются широким множеством пользователей, преследующих различные цели. Например, система ввода заказов может использоваться потребителями, являющимися в то же время сотрудниками отдела продаж, работа которых состоит в получении информации о состоянии нескольких поступивших заказов. Программа обработки заказов задействует различные части системы ввода заказов с целью получения информации о состоянии заказа и изменения части заказа. Программа обработки кредитных карточек обрабатывает исключения, которые генерируются модулем автоматизированной обработки кредитных карточек, являющимся частью системы ввода заказов. Для выполнения всего сценария в среде многопользовательского приложения требуется значительная работа по предварительному планированию, логической организации и установке терминалов, линий связи, начальной файловой структуры и координированию действий нескольких тестировщиков.

## Жизненный цикл дефекта

Чтобы можно было избавиться от блокировок при тестировании, для разработчиков программного обеспечения предельно важно исправлять программные ошибки как можно быстрее на этапах динамического тестирования. Как правило, блокировки при тестировании вызываются логическими ошибками, которые препятствуют обработке тестовых данных соответствующими строками кода. На ранних этапах динамического тестирования вполне естественно, что тестировщики и разработчики трудятся в тесном содружестве, обнаруживая и исправляя программные ошибки. Но на последних этапах тестирования или при работе над более крупными проектами роль координатора, контролера и расстановщика приоритетов играет группа контроля за внесением изменений (change control review board — CCRB), что иллюстрируется рис. 7.3. На этом рисунке показан жизненный цикл программной ошибки во время динамического тестирования. Когда программный модуль готов к тестированию, ведущий разработчик, тесно взаимодействующий с диспетчером конфигурации программного обеспечения (ДКПО), выполняет операцию сборки. В результате создается выполняемая версия программного продукта, и исходный код обновляется до новой версии.

Персонал группы тестирования отвечает за установку испытательной среды для данной выполняемой программы, включая создание компьютерных систем, операционных систем, баз данных, коммуникационных линий и т.п. Кроме того, этот персонал выбирает подходящий набор тестовых случаев. При выборе тестовых случаев учитывается природа и размещение в коде программных ошибок, обнаруженных модулем. В соответствии со сценариями тестирования специалист по тестированию исследует программу и последовательно сравнивает промежуточные результаты с результатами тестирования, которые задокументированы в тестовом случае. Несовпадение реальных и ожидаемых результатов свидетельствует либо о необходимости обновления ожидаемых результатов тестового случая реальными результатами, либо об обнаружении еще одной ошибки. После прогона выбранного набора тестовых случаев по отношению к создаваемой версии выполняется регрессивное тестирова-

ние с целью выяснения, не разрушили ли последние изменения какую-либо из функций, ранее успешно работавших. Характеристики любых новых ошибок вносятся в базу данных модели надежности и в отчет об ошибках, после чего цикл процесса отладки продолжается.

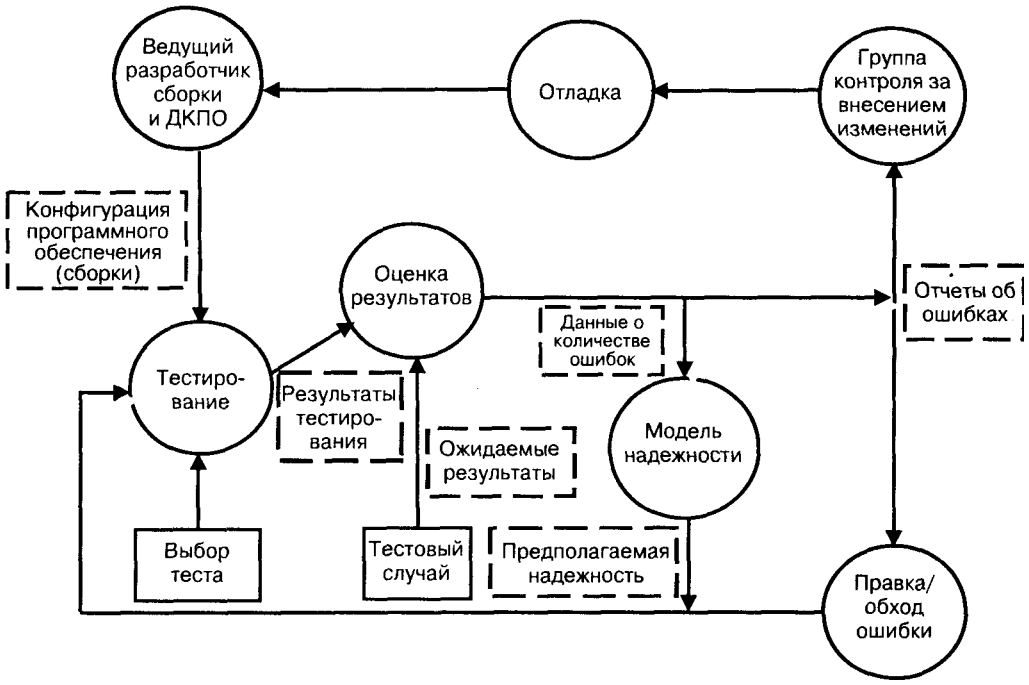



***Рис. 7.3. Взаимодействие разработчика и тестировщика: жизненный цикл программной ошибки.***

На рис. 7.3 показан также обходной путь для продолжения тестирования этой же версии. Он пролегает через процесс правки/обхода ошибки. Группе тестирования может потребоваться разделить тестовые случаи, которые все же должны прогоняться для текущей версии, на те, что столкнутся с этой же ошибкой, и на те, которые не должны быть подвержены ее воздействию. Это позволяет выполнять тестирование до тех пор, пока дефект не будет исправлен в будущих версиях. Иногда для продолжения тестирования требуется использование правки (patch) исполняемого модуля. В случае возникновения блокировки во время тестирования приоритет исправления ошибки повышается до наивысшего уровня. Тем самым задача немедленного исправления ошибки становится первоочередной.

То, что группа тестирования занята динамическим тестированием, не препятствует реализации отдельными членами группы дополнительных технологий статического тестирования. Все технологии статического тестирования полностью применимы во время выполнения динамического тестирования. Например, результаты ряда динамических тестов могут сравниваться с ожидаемыми результатами. Эта сравнительная оценка может быть автоматизирована или же в ходе ее выполнения может

использоваться процесс статического тестирования, аналогичный экспертной оценке. В нескольких компаниях было установлено, что некоторые из членов их групп тестирования не применяют адекватные технологии статического тестирования при анализе результатов. В этих компаниях говорят, что многие из специалистов, занятых динамическим тестированием, обладают менталитетом, который можно характеризовать следующим высказыванием: "Что ж, этот тест действительно не привел к отказу этого компьютера!" Подобный менталитет позволяет ошибкам оставаться незамеченными во время выполнения динамического тестирования, несмотря на то, что сами по себе тестовые случаи обеспечивают возможность обнаружения и ранее неизвестных ошибок.

Похоже, что менталитет наподобие "... это не привело к отказу компьютера" преобладает также среди персонала, занимающегося динамическим тестированием продуктов, в которых используется обмен данными через модемы. Обмен данными посредством модемов имеет долгую историю, и приобрел репутацию сопряженного с такими сбоями, как генерация сигналов занятости, несоответствие скорости передачи данных отвечающего и запрашивающего модемов или даже ошибки из категории "прочие", которыми тестировщики склонны объяснять любое беспричинное зависание отвечающего модема. Не сообщая о подобных ошибках, тестировщики подвергают свою компанию риску продолжать поставлять тестируемый продукт с неустраненными коммуникационными ошибками. Предположим, что подобный продукт был продан и установлен в нескольких помещениях больницы. Эти загадочные сбои типа "отвечающий модем просто зависает без видимой причины" будут как раз такими сбоями, которые становятся основной причиной нареканий со стороны сотрудников больницы, совершенно справедливо полагающих, что модемы должны немедленно обеспечивать соответствующую скорость обмена данными и соединение (при отсутствии сигнала занятости). Такие сбои не только становятся причиной обращения в службы технической поддержки больницы, телефонной компании и фирмы-изготовителя модема, но зачастую служат причиной претензий в адрес компании, которая выпустила или продала программный продукт.

## Формальные этапы тестирования

Динамическое тестирование может начинаться с этапа тестирования модулей и завершаться приемочными испытаниями. Стратегии динамического тестирования делятся на две категории: а именно на тестирование "черного ящика" и тестирование "белого ящика". Как правило, стратегия тестирования "белого ящика" применяется на этапах тестирования модулей и комплексных испытаний, в то время как стратегия тестирования "черного ящика" — на этапах системного тестирования и приемочных испытаний.

Цель тестирования модулей состоит в том, чтобы убедиться в реализации в коде всех запроектированных функций и в устойчивости воспроизведения общих возможностей и непротиворечивости выполнения программного модуля. При тестировании модулей для управления тестом во время выполнения часто применяются драйверы и заглушки, в то время как при комплексных испытаниях для управления комплексными функциями, образованными несколькими модулями или подсистемами, как правило, используется GUI-интерфейс.

Цель, комплексных испытаний является необходимость убедиться в соответствии по входным и выходным параметрам всех интерфейсов между модулями или подсистемами, а также в правильности других совместно используемых или передаваемых данных, таких как, например, записи и поля баз данных, совместно используемые экраны GUI-интерфейсов или разделяемые коммуникационные магистрали.

Системное тестирование может начинаться непосредственно после завершения комплексных испытаний всех модулей или, по крайней мере, тех из них, которые образуют основную подсистему. Системное тестирование должно фокусироваться на глобальных вопросах производительности, масштабируемости, пригодности к установке, устойчивости к воздействиям окружающей среды и общей применимости конечными пользователями.

В ходе приемочных испытаний версии программных продуктов, являющиеся кандидатами для поставки на рынок, исследуются с целью определения их приемлемости для конечных пользователей. В случае применения методологии быстрого тестирования для выполнения всех этих этапов жизненного цикла тестирования требуется все меньше и меньше времени, поскольку большая часть ошибок обнаруживается **и** устраняется на ранних этапах разработки.

# Обязанности членов команды тестирования

Специалисты по тестированию выполняют подготовку к тестированию, прогоняют тесты и докладывают о результатах тестирования. Существует много должностей, необходимых для обеспечения полноценной работы команды тестирования. Эти должности, особенно в небольших проектах, могут занимать лица, обладающие подготовкой в нескольких областях. Краткий перечень должностей приведен в таблице 7.2.

**Таблица 7.2 Обязанности членов команды тестирования**

| Выполняемая работа/должность | Обязанности |
|---|---|
| Обеспечение качества | Предоставляет перечень и подробные определения всех действий по тестированию, включая шаблоны и примеры. |
| | Обеспечивает инструктаж и тренировку по созданию планов тестирования и тестовых случаев. |
| | Обеспечивает текущий мониторинг и поддержку во время проведения тестирования с учетом точного распределения ролей в каждом из планов тестирования. |
| Аудит и управление | Обеспечивает ввод дополнительных условий, которые должны оказывать влияние на управление тестированием, и выполняет другие функции, связанные с аудитом. |
| | Предоставляет обзоры, касающиеся соответствия стандартам по аудиту. |

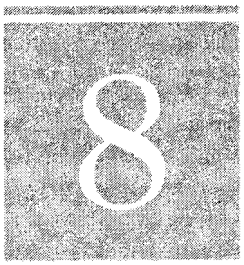
***Окончание табл. 7.2***

| **Выполняемая работа/должность** | **Обязанности** |
|---|---|
| Разработчик программного обеспечения | Предоставляет документацию по тестированию модулей. |
| | Оказывает тестировщикам помощь в создании тестовых случаев для проведения испытаний основных компонентов, комплексных, системных и приемочных испытаний. |
| | Осуществляет руководство разрешением проблем во время выполнения тестирования. |
| | Выполняет тестирование модулей. |
| Бизнес-аналитик | Создает план приемочных испытаний. |
| | Создает тестовые случаи для проведения приемочных испытаний. |
| | В содружестве с тестировщиком создает тестовые случаи для проведения тестирования основных компонентов. |
| | Проводит приемочные испытания. |
| | Просматривает и утверждает критерии приемки. |
| Разработчик тестов | Создает тестовые случаи для проведения тестирования основных компонентов, комплексных и системных испытаний. |
| | Проводит комплексные и системные испытания. |
| | Во время проведения приемочных испытаний предоставляет техническую поддержку с целью разрешения возникающих проблем и вопросов. |
| Тестировщик | Прогоняет тестовые случаи. |
| | Записывает реальные результаты. |
| | Заполняет отчеты о возникших во время тестирования проблемах. |
| | Проверяет устранение обнаруженных проблем. |
| Отладчик | Обеспечивает устранение обнаруженных проблем. |
| | Проверяет действительное наличие обнаруженных проблем. |
| | Обеспечивает устранение всех аспектов обнаруженной проблемы, в том числе исправление документации для пользователя. |
| Менеджер-тестирования | Обеспечивает создание инфраструктуры тестирования, в том числе управление тестированием и обеспечение инструментальными средствами для его проведения. |
| | Осуществляет координирование/управление всеми действиями по тестированию. |
| | По мере необходимости определяет приоритетность устранения проблем. |
| | Докладывает о ходе тестирования руководству проекта и руководству службы по работе с клиентами/пользователями. |
| | Отслеживает соответствие между реальными и плановыми сроками выполняемых работ. |
| Администратор тестирования | Регистрирует отчеты о возникающих в ходе тестирования проблемах. |
| | Подготавливает ежедневные сводки о состоянии выполнения работ, сравнивая их выполнение с планом и с общим объемом работ по тестированию. |

## Что дальше

**В** главах 8 и 9 подробно рассматриваются технологии статического тестирования. Глава 10 посвящена технологиям динамического тестирования. В главах 11 и 12 освещены показатели тестирования и модели оценки затрат на тестирование. Вот названия этих глав:

8. Совместная разработка требований к приложению (JAR): метод выработки требований с применением быстрого тестирования
9. Технологии статического тестирования и советы
10. Технологии и советы по динамическому тестированию
11. Разработка и использование показателей тестирования: моделирование и прогнозирование ошибок
12. Технологии оценки трудозатрат на тестирование и советы



# Совместная разработка требований к приложению (JAR): метод выработки требований с применением быстрого тестирования

**Темы, рассматриваемые в главе:**

- Методология **JAR**
- Роль специалистов по тестированию в процессе **JAR**
- Резюме

Одно из основных условий успешной разработки программного обеспечения — достижение понимания требований клиентов. Как было показано в главе 2, четкое определение технических требований служит отправной точкой эффективного процесса разработки программного обеспечения.

Существуют различные методы, призванные улучшить обмен информацией между клиентом и командой разработчиков. Один класс методов, используемых для выработки технических требований, называется технологией ускоренной разработки спецификации приложения (fast application specification techniques, FAST), которая кратко описывалась в главе 2. Совместная разработка требований к приложению (Joint Application Requirements, JAR) — это технология FAST, которая предназначена для обеспечения полной интеграции статического тестирования с процессом выработки требований. Интеграция статического тестирования в процесс JAR в виде его составной части достигается за счет выполнения действий, известных под названием совершенной поддержки. Совершенная поддержка — это организованный способ интерактивного анализа и пересмотра выработанных требований. Действия, связанные с совершенной поддержкой, описаны далее в этой главе.

Результатом JAR-сеанса является набор тщательно проанализированных требований, которые согласованы с пользователями конечного продукта или их представи-

телями. Требования, выбранные в JAR-сеансе, включают в себя как функциональные, так и подходящие нефункциональные требования. Например, сюда может входить календарный план поставки продукта по этапам.

## Методология JAR

Участвующие в JAR-сеансе образуют комиссию, составленную из представителей команды разработчиков, нескольких пользователей, знакомых с различными функциональными возможностями, которые должны присутствовать в конечном продукте, представителя клиента/покупателя и председателя комиссии JAR. Как правило, для проведения заседаний комиссии JAR на одну-две недели выделяется просторный конференц-зал. Одна из возможных схем расположения мест в конференц-зале с учетом ролей приглашенных для участия в JAR людей показана на рис. 8.1.

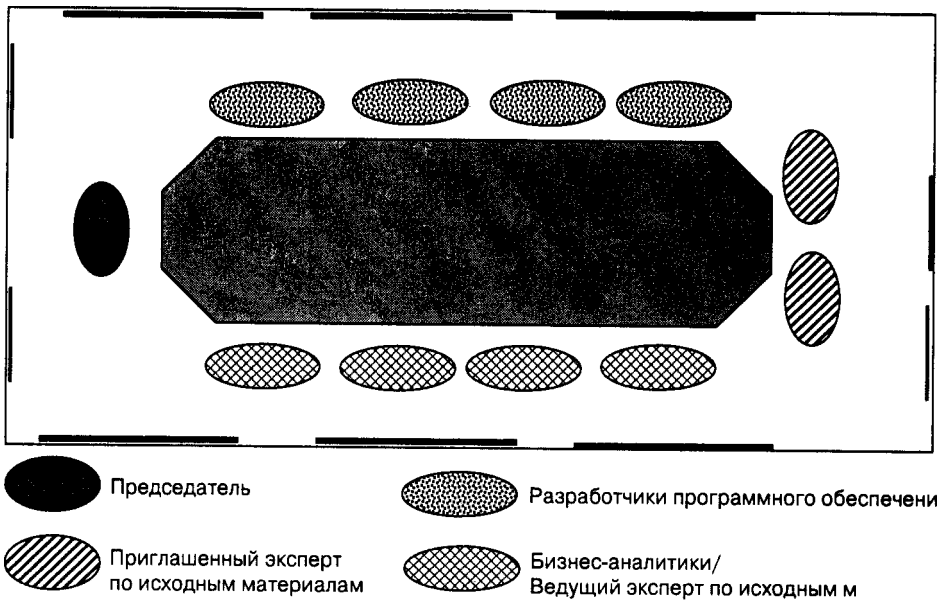



*Рис. 8.1. Размещение основных участников в помещении для проведения **JAR.***

Обратите внимание, что в состав комиссии не входят ни менеджеры, ни вице-президенты, ни представители отдела обеспечения качества, ни представители отдела снабжения, ни представители отдела поставки. Если кто-то из представителей этих подразделений появляется на одном из заседаний комиссии, председатель должен попросить их покинуть зал на период обсуждения вопросов совершенной поддержки, которые будут описаны позже. Члены комиссии не должны пропускать заседания и их не должны отвлекать сообщениями или телефонными звонками.

Роль председателя состоит в постоянно удерживать в центре внимания команды три следующих задачи: выработка требований, четкая формулировка требований и уточнение требований. Накануне заседания комиссии JAR председатель должен провести установочное совещание для ознакомления членов комиссии со стоящими перед ней целями и правилами работы. В ходе этого установочного совещания члены

комиссии должны дать свое согласие действовать в соответствии со следующими правилами:

- Не допускать личных нападок. Например, не пренебрегать и не дискредитировать идеи, высказанные другими участниками.
- Помнить, что одновременно может высказываться только один человек.
- Прежде чем предлагать какую-то идею, подумать о том, какова ее ценность, и окажет ли она влияние на конечный продукт.
- Тщательно формулировать свои предложения, и после их анализа документировать их в заключительном документе.
- Задавать вопросы только председателю.
- Подавать председателю сигнал о необходимости перерыва в заседании (например, скрестив руки), принятый в комиссии на случай возникновения непредвиденных ситуаций/перерывов (например, при необходимости перерыва для отдыха или если был замечен какой-либо личный выпад).
- Во время заседания комиссии JAR не допускать проявления никаких внешних отвлекающих факторов (сообщений электронной почты, вызовов по пейджеру или телефону).
- Присутствовать на заседаниях, быть внимательным, творческим и полезным.

Председатель знакомит членов комиссии с целями и ограничениями проекта по созданию программного обеспечения, а также определяет цели работы комиссии JAR, в числе которых:

- Сбор сведений о потребностях и ожиданиях пользователей, а также определение показателей эффективности.
- Анализ потребностей и ожиданий пользователей для выработки словесного описания функционирования системы и отдельных функций, которые должны выполняться продуктом.
- Выработка требований по поставке и модернизации продуктов, которые должны устанавливаться и модернизироваться на рабочих местах пользователей.
- Разработка формулировок всех разделов требований, которые должны быть освещены в итоговом документе.
- Получение от клиента и пользователей подтверждений того, что формулировки требований удовлетворяют их потребности и ожидания.
- Составление поэтапного графика поставки продукта с расширяющимся набором возможностей с целью удовлетворения текущих и долговременных потребностей и ожиданий пользователей, а также определение показателей эффективности этих поэтапных поставок.

В состав комиссии должны входить опытные эксперты по исходным материалам (source matter expert, SME), обладающие глубокими познаниями в области бизнес-правил, областей применения и потоков данных, которые должны учитываться во время разработки новой системы. Приходящие эксперты по исходным материалам приглашаются на интервью, посвященные выработке требований, каждое из которых длится около 1,5 часов. На эти интервью могут приглашаться до двадцати экс-

пертов, по двое одновременно. По окончания каждого интервью комиссии выделяется время на документирование требований, которые были выработаны на основе доклада эксперта, и для отражения мероприятий по совершенной поддержке на плакатах, которые применяются для фиксации требований.

Типовой календарный график работы комиссии JAR типа "когда, кто и почему" показан в табл. 8.1. В этом примере представлены семь отдельных заседаний: установочное, три дня интервью с экспертами по исходным материалам, два рабочих дня и один день подготовки материалов для формальной документации. Поскольку в ходе каждого заседания, посвященного интервью с экспертами, беседа проводится с двумя из них, в этом примере в заседаниях комиссии JAR всего принимают участие 24 эксперта. Прежде чем начнутся рабочие заседания, комиссии может потребоваться полдня для реорганизации настенных плакатов. С целью минимизации ощущения избыточности может потребоваться объединение аналогичных плакатов. Кроме того, может возникнуть необходимость определить приоритеты требований в двух наборах, например, разделение требований на те, что могут быть реализованы на первом этапе представления проекта, и те, которые должны быть реализованы на втором этапе. Поскольку каждое рабочее заседание проходит в среде, подобной библиотечной, на каждое из 6 рабочих заседаний приглашаются только по 4 эксперта по исходным материалам. Кроме экспертов по исходным материалам, на рабочие заседания в качестве наблюдателей могут быть приглашены несколько лиц, чье служебное положение не позволяет им входить в состав комиссии. Между рабочими заседаниями члены комиссии выполняют процедуры совершенной поддержки для включения комментариев экспертов по исходным материалам в основной набор плакатов.

**Таблица 8.1.,Типовой календарный график заседаний комиссии JAR**

| Когда | Кто | Тема | В соответствии с графиком, составленным председателем |
|---|---|---|---|
| Вводное заседание | Члены комиссии и заинтересованные лица | Инструктаж | |
| 1-й день | Члены комиссии | 2 заседания с участием экспертов по исходным материалам | |
| 2-й день | Члены комиссии | 4 заседания с участием экспертов по исходным материалам | Совершенная поддержка |
| 3-й день | Члены комиссии | 4 заседания с участием экспертов по исходным материалам | Совершенная поддержка |
| 4-й день | Члены комиссии и группы экспертов по исходным материалам | 2 заседания с участием экспертов по исходным материалам<br>2 рабочих заседания | |
| 5-й день | Члены комиссии и группы экспертов по исходным материалам | 4 рабочих заседания | Совершенная поддержка |
| 6 день | Члены подкомиссии | Организация и встреча с оформителем технической документации | Совершенная поддержка |

Опытный председатель комиссии JAR запасется, по меньшей мере, несколькими сотнями чистых листов бумаги формата 8,5x11 дюймов (A4), десятком катушек липкой ленты, несколькими десятками разноцветных шариковых ручек, подставкой для документов, разноцветными маркерами, ножницами, наклейками и целлофановой лентой. Эти материалы будут использоваться во время подготовки и поддержки настенных плакатов, на которых отражаются требования.

Расположение плакатов на стенах конференц-зала должно соответствовать структуре документа описания требований. На заглавных листах могут быть напечатаны названия разделов этого документа. Они могут подвешиваться на верхней части стены, через равные промежутки, и должны располагаться по часовой стрелке в соответствии со структурой документа.

Каждый плакат будет иметь соответствующий заглавный лист. На шаблоне плаката, приведенном на рис. 8.2, показано, как вводить требование и с помощью клейкой ленты прикреплять этот лист под соответствующим заглавным листом. Инструктаж по созданию подобного плаката из чистого листа бумаги, должен проводится на установочном заседании. Шаблон не должен содержать заранее впечатанной информации, поскольку это будет препятствовать спонтанности, которую автор должен проявлять во время заполнения плаката с требованиями. Прикреплять листы формата 8,5x11 дюймов на стенку следует только за верхние углы, соблюдая альбомную ориентацию. Это позволит при необходимости поднимать плакаты вверх и перемещать их на другую часть стенки, если потребуется сгруппировать их с другим требованием.

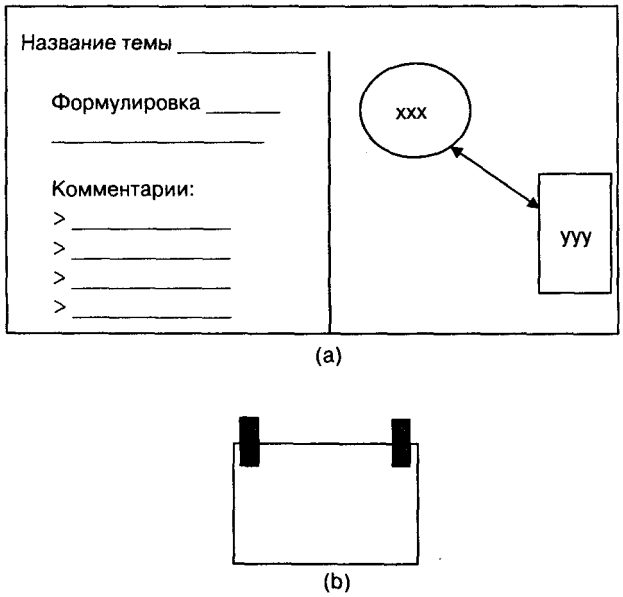



*Рис. 8.2. (a) Шаблоны плакатов и (b) технология прикрепления к стенке.*

Строка названия темы должна совпадать с заглавным листом, помещенным на стенку. Если плакат перемещается в набор плакатов, расположенных под другим заглавным листом, эта строка должна быть изменена. Формулировка — это краткое изложение формулируемого требования. Она может представлять собой уточнение области действия требования, описание применяемых интерфейсов или касаться любых других вопросов, требующих прояснения. В правой части плаката размещается диаграмма, в которой требование представляется в графическом виде. В одном случае это может быть диаграмма взаимодействий, в которой показаны соединения между элементами. В другом случае это могут быть фигурки беседующих людей, изображающие процесс опроса вручную. В третьем — диаграмма взаимодействия одной подсистемы компьютерной программы с другой через базу данных. Обычно под графической диаграммой помещается ее название.

Выполняя действия по совершенной поддержке, членам комиссии приходится часто подходить к стенам помещения. Эти действия не отражаются в календарном графике JAR, поэтому по мере необходимости председатель должен время от времени требовать от членов комиссии, чтобы они подходили к стенам, внимательно изучали каждый плакат и фиксировали предложения по изменениям, наклеенные непосредственно на плакаты. Эта форма ревизии тесно связана с процессом выработки требований. Обоснование написания наклеек приведено на рис. 8.3. Они служат для того, чтобы всем участникам стало понятно, что пора оказать помощь в обнаружении ошибок в требованиях или же четче сформулировать требования, а то и вовсе изменить их расположение на стенке. На этом этапе совершенной поддержки изменения в самих плакатах не допускаются.

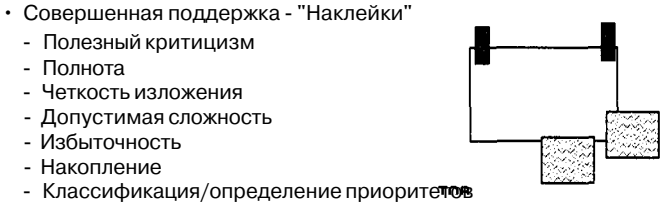



*Рис. 8.3. Механизмы совершенной поддержки.*

В процессе выполнения этих действия стенка обычно приобретает цвет приколотых примечаний, поэтому председатель приостанавливает выполнение всех действий, дабы перейти к следующему этапу процесса совершенной поддержки. Первоначальный владелец плаката, на котором имеются наклейки, выполняет указанные ими обновления, периодически восклицая: "Чье это предложение по такому-то требованию?" Часто автор наклейки и автор плаката обмениваются мнениями с целью выработки приемлемого решения. Этот процесс обеспечивает достижение двух важных целей: формулировки требований улучшаются в смысле четкости выражений и независимости от других требований, а вклад всех членов комиссии и экспертов по исходным материалам реализуется совершенно естественным образом. Подобная совместная работа благотворно сказывается на проекте, гарантируя соответствие продукта реальным требованиям пользователей. Во время создания программного продукта будет возникать множество ситуаций, когда команда разработки должна взаи-

модействовать с группой пользователей или с экспертами по исходным материалам. Такие ситуации отражаются в следующем списке:

- Организация работы рабочей группы управления взаимодействием
- Организация работы рабочей группы технического управления
- Выполнение промежуточных пересмотров программы
- Работа с опросными листами, интервью, эксплуатационными сценариями, полученными от конечных пользователей (случаи использования на языке UML)
- Создание прототипов и моделей
- Наблюдение за существующей системой, средами и конфигурациями технологических процессов
- Принятие или заимствование решений и осуществление последующего выбора
- Альфа-тестирование
- Бета-тестирование

Это взаимодействие будет более плодотворным и качественным, если все члены этой сборной команды научатся работать вместе и будут знать основные требования. Результатом работы комиссии JAR должен быть набор плакатов, содержащих функциональные и некоторые нефункциональные требования, предъявляемые к новому проекту. Напряженная работа участников JAR по уточнению формулировок, графического материала, приоритетов и заголовка каждого требования позволяет опытному техническому оформителю без особого труда заполнить шаблон для создания завершенного документа с требованиями.

#### ПРИМЕР: УСПЕШНАЯ РАЗРАБОТКА JAR (ГЭРИ КОББ (GARY COBB))

Несколько лет назад мне довелось столкнуться с разработкой очень важного проекта по электроиной торговле, календарный план которого был очень плотным. Этот проект был явным кандидатом на применение технологий быстрого тестирования. Основная задача формулировалась как быстрая разработка интерфейса программирования приложений (API) между системой заказов товаров по Internet и стандартной системой размещения заказов компании. Выделенные руководством денежные средства предполагали ведение разработки шестью разработчиками, включая руководителя/ведущего разработчика, в течение 3 месяцев. Руководитель проекта выделил б недель из 16-недельного календарного плана на документирование требований и поручил эту работу двум разработчикам, в то время как остальные должны были приступить к созданию прототипа API. Тогда я выступил с заявлением, что если бы в проекте был использован процесс JAR, то я смог бы предоставить полностью протестированный документ описания требований за 2 недели — на целый месяц раньше планового срока. При этом в нем были бы полностью учтены пожелания пользователей, и было бы меньше ошибок, чем при выполнении 6-недельной работы, за которую они были готовы взяться.

В четверг руководитель проекта и его подчиненные согласились принять мое предложение по применению JAR, и пятницу я начал с проведения вводного собрания, во время которого группа составила список из 28 экспертов по исходным материалам. Во время вводного собрания было выбрано также рабочее помещение, а четыре наиболее знающих эксперта были рекомендованы как члены команды на следующую неделю (ежедневно с 8:00 до 17:00). Были расписаны также часы бесед с остальными 24-мя экспертами (12 бесед с двумя экспертами ежедневно, с понедельника по среду следующей недели)

**Легкий утренний завтрак и послеобеденные освежающие напитки должны были подаваться в общем помещении, однако промежуток с полудня до 13:00 оставался свободным, чтобы члены группы могли пообедать и ответить на сообщения.**

**Утром в понедельник члены комиссии мучались над созданием плакатов для ряда фундаментальных требований. Ряд споров велось по поводу элементов архитектуры системы, эксплуатационных характеристик и применимости, производительности и требований к внешним коммуникациям. Затем после полудня состоялись два первых интервью с экспертами по исходным материалам. По завершении интервью, во вторник состоялась первая половина заседания по совершенной поддержке. В результате вся стена окрасилась в розовый закатный цвет, поскольку председатель располагал только розовыми наклейками. До проведения первого интервью с экспертом по исходным материалам в среду авторы плакатов обработали поставленные вопросы и стена вновь стала белой. За вторую половину среды, в процессе подготовки к первому рабочему заседанию, которое должно было состояться в четверг, сеанс совершенной поддержки был завершен, и члены комиссии сложили часть плакатов под поручнями кресел, тем самым признав, что эти требования имеют более низкий приоритет и должны быть выполнены позже. Одновременно один из членов комиссии завершил уточнение глоссария. В полдень четверга двум менеджерам было предложено оставить свои замечания на стене с пустыми розовыми наклейками. Члены комиссии были весьма горды похвальными отзывами, оставленными обоими менеджерами.**

**К пятнице помещение JAR было проветрено, а комиссия, за исключением председателя и руководителя проекта, которые остались для беседы с техническим оформителем, была распущена. Руководитель проекта согласился к понедельнику пересмотреть календарный план с учетом более раннего начала проектирования. Технический оформитель согласился ко вторнику подготовить черновик документа с требованиями и новый календарный план разработки проекта.**

**Во время этой разработки JAR были запланированы три последовательных поставки версий программы, причем первая должна была быть создана по истечении первых трех месяцев. Эта версия должна была, как и планировалось ранее, передавать Internet-заказы в систему размещения заказов. Второй этап, разработка которого должна была выполняться параллельно первому, ориентировался на создание ASP-системы, запускаемой во время формирования Internet-заказа, как средства предотвращения попадания неверно оформленных заказов в систему размещения заказов. На третьем этапе, разрабатываемом параллельно с первым и вторым этапами, планировалась разработка Web-системы отслеживания заказов, предназначенной для автоматизации получения информации о состоянии заказов Internet-клиентов. Поставки версий программы на всех этих этапах выполнялись своевременно и были с радостью встречены перегруженным работой персоналом отдела обработки заказов, котором до этого приходилось вводить и обрабатывать заказы вручную.**

**Наиболее примечательное замечание, впоследствии услышанное мною от одного из обработчиков заказов, звучало так: "После получения программы члены этой команды разработчиков учили нас выполнять работу с помощью данного программного обеспечения, но даже сами разработчики не смогли заставить программу работать во время обучения. Когда они поставили программу, мы и сами знали, как ее использовать, поскольку она полностью соответствовала нашим потребностям. Удивляло лишь то, что в программе было очень мало ошибок, и что на всех этапах поставка выполнялась точно по графику." Отойдя в сторону, я сказал себе: "Да! Вот вам результат одновременного выполнения разработки и тестирования!" Иначе говоря, таков результат быстрого тестирование в действии.**

# Роль специалистов по тестированию в процессе JAR

**В** процессе JAR специалисты по тестированию играют три роли. Они (1) участвуют в совершенной поддержке, (2) знакомятся с требованиями к тестированию, которые должны стать обоснованием требований по выделению ресурсов для персонала проведения испытаний и (3) во время интервью с экспертами по исходным материалам задают вопросы относительно используемых в настоящее время процедур тестирования. Один из разделов документа с требованиями должен быть посвящен требованиям к тестированию продукта, и именно специалист по тестированию должен первым ратовать за то, чтобы этот раздел был максимально полным и точно сформулированным. Вполне вероятно, что впоследствии он окажется тем тестировщиком, которому придется преобразовывать эти требования к тестированию в требования к ресурсам по обеспечению тестирования, общий план тестирования и множество тестовых случаев со сценариями и результатами тестирования. Во время интервью с экспертами по исходным материалам тестировщик должен прояснить каждый из перечисленных ниже вопросов:

- Расскажите о режимах, приводящих к отказам используемых систем, наиболее часто отмечаемые пользователями.
- Пожалуйста, опишите методику взаимодействия пользователей с группой разработки во время документирования, исправления и тестирования ошибок в используемой системе.
- Опишите, как пользователи осуществляют переход к новой версии, выпущенной группой разработчиков.
- Назовите некоторые из стандартов, которые пользователи желают выдвинуть разработчикам нового продукта.
- Пожалуйста, помогите описать некоторые платформы и языки, в сочетании с которыми новый продукт должен функционировать, а также любые конкретные соображения по поводу окружающей среды компьютерных систем, включая доступность для обслуживания, ремонта и т.п.

При рассмотрении тестирования на этапе выработки требований в рамках жизненного цикла разработки следует помнить, что во время JAR-сеанса выбираются и помещаются в итоговый документ как функциональные, так и нефункциональные требования, при этом они должны быть максимально полными и четко сформулированными. Но они не являются единственными, которые должны быть удовлетворены во время разработки, равно как и единственными, которые должны тестироваться. Ниже приведен перечень ряда дополнительных требований, которые должны быть учтены персоналом разработки:

- **Стандарты разработки проекта.** Это требования к процессам и процедурам, которым персонал разработки должен будет следовать на этапах разработки проекта. Весьма вероятно, что эти установки и процедуры будут стандартными для всех разрабатываемых проектов, поэтому персонал тестирования должен располагать инструментальными средствами и тестовыми случаями, которые позволят проверить каждый продукт на предмет соответствия этим требованиям.

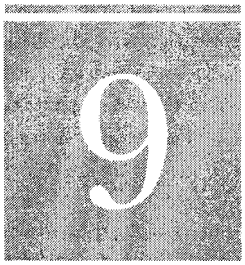
- **Производные требования.** В течение JAR-сеанса эксперт по исходным материалам может заявить, что новый продукт должен быть выполняемой программой, запускаемой на одном из сетевых клиентских компьютеров, работающих при нормальных рабочих условиях окружающей среды. Видя это требование, проектировщики программного обеспечения могут включить в код параметр для комнатной температуры и поискать значения верхней и нижней границ температуры в стандарте, который определяет нормальные рабочие условия окружающей среды. При создании тестового случая для проверки соответствия исходному требованию, чтобы воспроизвести испытательные условия окружающей среды для тестируемого продукта, специалистам по тестированию придется обратиться к производному требованию, выдвинутому проектировщиками программного обеспечения.
- **Требования к интерфейсу.** Занимаясь реализацией нового продукта, проектировщики программного обеспечения могут принимать решения по созданию или приобретению тех или иных компонентов. Чтобы можно было взаимодействовать с приобретенными подсистемами, конструкция должна удовлетворять дополнительным нефункциональным требованиям. При создании тестовых случаев для тестирования интерфейсов на предмет их соответствия спецификациям тестировщики будут применять тесты к интерфейсу этой подсистемы так, как это определено в документации, поставляемой вместе с подсистемой.

# Резюме

Подводя итоги сказанному в этом разделе, который был посвящен объединению тестирования с выработкой требований, следует отметить, что в действительности JAR — всего лишь один из множества методов выработки требований. Основная идея состоит в том, что вся команда разработки оценит роль тестирования только тогда, когда с самого начала оно будет интегрировано в процесс разработки. Разработчики смогут оценить преимущества того, что не придется проектировать и реализовать сложный программный продукт лишь для того, чтобы выяснить, что основные пользователи не в состоянии понять принципы его применения, а покупатели не желают расставаться со своими деньгами.

Тестировщики системы также оценят избавление от необходимости разрабатывать и реализовывать сложные тестовые случаи для этой сложной конструкции. Персонал отдела маркетинга оценит отсутствие необходимости конкурировать с более дешевым продуктом, появившимся на рынке раньше, поскольку он не содержит сложных компонентов. И, наконец, поэтапная поставка версий продуктов является общепринятой нормой в промышленности программного обеспечения, а сложные компоненты, исключенные из первоначальных требований, могут послужить реальным усовершенствованием, которое привлечет симпатии клиентов, если они будут добавлены и реализованы в форме существенной модернизации.

# Технологии статического тестирования и советы




**Темы, рассматриваемые в главе:**

- Цикломатическая сложность и ее взаимосвязь с выполнением тестирования
- Пример представления проекта модуля в виде графа
- Формальная оценка
- Применение контрольных перечней
- Аудит
- Инспекции/критический анализ/экспертные оценки
- Распределение ролей и обязанностей в группе выполнения инспекций
- Отчетность о процессе выполнения инспекций
- Показатели процесса инспекции
- Использование электронной почты или другого электронного приложения для ускорения процесса инспекции
- Формальная верификация
- Языки на основе спецификаций
- Автоматизированное доказательство теорем
- Средства автоматизации тестирования
- Прослеживаемость требований
- Программа контроля единиц измерения физических величин
- Символьное выполнение
- Листинги перекрестных ссылок
- Программы улучшенной печати
- Средства сравнения версий
- Тестирование алгоритмов
- Диспетчер тестирования
- Базы данных материалов совместного использования
- Резюме

Существует множество технологий, в которых используется анализ архитектуры и структуры программных разработок. В этой главе описан ряд таких технологий. В соответствии с концепцией быстрого тестирования эти технологии должны применяться на максимально ранних этапах жизненного цикла разработки, и большинство из них может использоваться еще до начала создания каких-либо кодов. Известно, что высокая сложность неизбежно приводит к большому количеству ошибок, поэтому в этой главе внимание уделяется, прежде всего, вопросам уменьшения сложности. Основной побочный эффект измерения цикломатической сложности состоит в том, что оно позволяет группе тестирования оценить количество прогонов программного объекта, которое потребуется для хотя бы однократного тестирования каждой ветви программы. Это может служить ориентиром как для планирования трудозатрат на тестирование, так и для создания упорядоченных наборов случаев использования.

Формальные оценки, инспекции и аудиты позволяют убедиться в том, что каждый из разработчиков придерживается наборов установок, процедур, руководств по стилю и других стандартов, используемых в организации. Кроме того, они являются эффективным средством обнаружения ошибок. В главе также рассматриваются технологии, которые должны применяться для организации и проведения оценок и инспекций.

# Цикломатическая сложность и ее взаимосвязь с выполнением тестирования

Наличие нескольких ветвей в программе существенно повышает вероятность ошибок, и поэтому тестировщики уделяют им особое внимание. Вполне естественно, что многие разработчики программного обеспечения считают операторы GOTO, т.е. безусловные переходы, конструкцией языка, которая затрудняет понимание кода. Оператор GOTO исключен из большинства структурированных языков программирования. Тем не менее, точки принятия решений являются обязательными конструкциями даже в таких языках. Например, и конструкция IF(...), и конструкция DO WHILE(...) содержит точку принятия решения.

В процедурных языках ветвь определяется как переход в процессе вычислений, начинающийся от точки входа в модуль или точки принятия решения и заканчивающийся точкой выхода из модуля или точкой принятия решения. В некоторых работах, посвященных этой теме, авторы называют ветвь DD-ветвью (от decision-to-decision path — ветвь "от-решения-до-решения"). Поэтому типичная конструкция IF-THEN-ELSE-ENDIF содержит две ветви, которые обычно называются ветвями THEN и ELSE, причем обе они завершаются ключевым словом ENDIF конструкции. В математике и в теории графов было показано, что граф структурированной программы является планарным (плоским). И наоборот, все неструктурированные программы имеют непланарные графы. Уже на заре программирования стало привычным для упрощения понимания логики вычислений вычерчивать линии, соединяющие точки принятия решений. Тогда же стал привычным отказ от попыток разобраться в программах, содержащих слишком много ветвей, и разбиение их на доступные для понимания фрагменты.

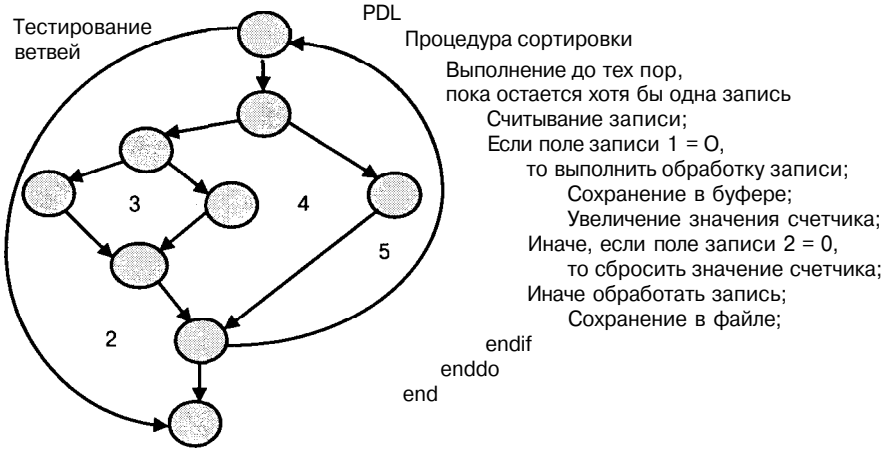
Вполне естественно, что проблема сложности программного обеспечения уже давно привлекает внимание испытателей. Руководители различных рангов весьма

довольны, когда представитель отдела тестирования заявляет: "Да, мы, по крайней мере, один раз протестировали каждую строку кода". Томас Маккейб (Thomas McCabe) и Чарльз Батлер (Charles Butler) [32] в качестве показателя определили числовое значение цикломатической сложности $V_e$ Это значение может быть вычислено по одному из трех способов:

1. Путем подсчета количества областей планарного графа, который представляет процесс структурированной программы.
2. Цикломатическая сложность ($V_g$) графа процесса $g$ определяется выражением $V=E- N + 2$, где $E$— количество ребер графа процесса, а $N$— количество узлов графа процесса.
3. Цикломатическая сложность ($V_g$) графа процесса $g$ определяется также выражением $V_g=P + 1$, где $P$ — количество предикатов (логических утверждений) в графе процесса.

# Пример представления проекта модуля в виде графа

Показанный на рис. 9.1 планарный граф делит плоскость на пять областей, пронумерованных цифрами от 1 до 5, поэтому его цикломатическая сложность $V=5$. Попытайтесь найти четыре предиката в коде и в соответствующем ему графе процесса. В заключение определите количество тестовых случаев, которые потребовались бы, по меньшей мере, для однократного тестирования каждой строки кода.




**Примечание:** каждая структурированная программа имеет направленный планарный граф.

***Рис. 9.1. Планарный граф программы, иллюстрирующий определение показателя цикломатической сложности.***

Значение цикломатической сложности для тестирования программ состоит в том, что *V* представляет собой минимальное количество независимых ветвей, которые должны быть проверены для тестирования всей программы. Испытатель должен быть в состоянии применить это свойство к графу "a", представленному в левой части рис. 9.2, и разработать минимальное количество тестовых случаев, которые должны быть выполнены для полного охвата всего приложения, соответствующего данному графу. При этом возникает ряд вопросов. Обеспечивают ли два приведенных тестовых случая полное тестирование всех ветвей? Нет ли какой-либо ветви, которая осталась неохваченной тестовыми случаями 1 и 2? И если да, то каким должен быть дополнительный тестовый случай?

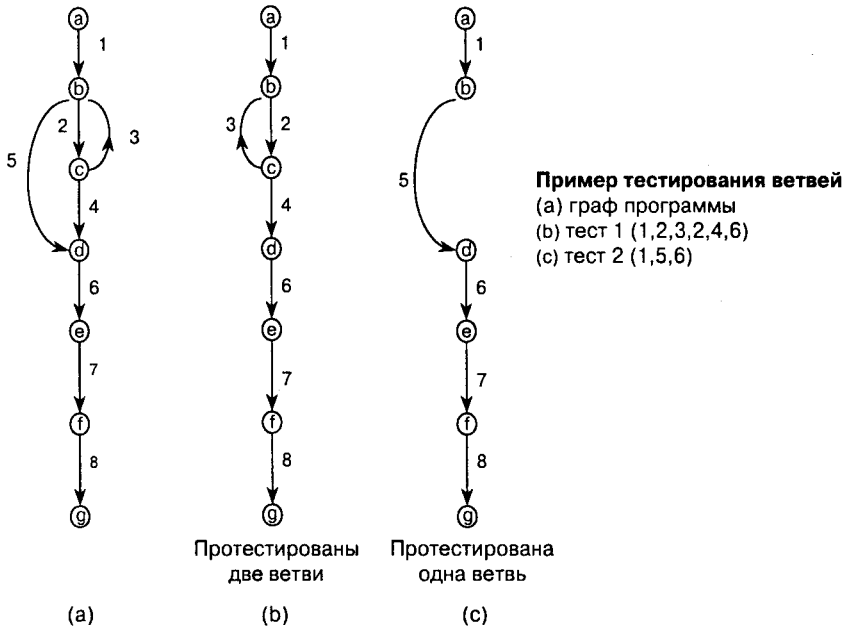



*Рис. 9.2. Определение минимального набора тестовых случаев на основе значения показателя цикломатической сложности.*

В [18] Питер Чейз Белфорд (Peter Chase Belford) описал, как в рамках одного из выполненных им проектов в корпорации Computer Science Corporation при помощи показателя цикломатической сложности удалось снизить стоимость проекта и одновременно повысить качество программного продукта за счет снижения сложности кода.

### ПРИМЕР: ПРОЕКТ CENTRAL FLOW CONTROL

Эти учебные примеры заимствованы из [18]. В проекте Central Flow Control (CFC) для документирования алгоритмов использовался псевдокод. Для языка разработки псевдокода (pseudocode design language, PDL) была создана программа синтаксического анализа, в которой был реализован счетчик цикломатической сложности.

**Стоимость разработки каждой модели и соответствующее ей значение показателя цикломатической сложности наносились на фафик, и для полученных данных определялась наиболее подходящая S-образная кривая. S-образная кривая Белфорда показана на рис. 9.3. Из приведенного рисунка видно, что существовал диапазон цикломатических значений модулей — например, вблизи линейного участка кривой, — которые отражали приемлемую стоимость разработки, выраженную в часах. В то же время другая область представляла неприемлемое возрастание стоимости. В ходе разработки проекта CFC постоянно выполнялось повторное проектирование модулей, для которых показатель цикломатической сложности PDL превышал 30.**

**На основе анализа процесса разработки этой профаммы Белфорд пришел к следующим выводам:**

- **Лучше использовать меньшие модули.**

  **Сложность системы разработки программного обеспечения лучше измерять количеством ветвей принятия решений, чем количеством выполняемых операторов в исходном коде.**

  **Использование современных методов программирования способствует повышению степени понятности и надежности системы, а также делает процесс разработки более управляемым.**

  **Надежность программного обеспечения повышается с увеличением времени, которое тратится на проектирование.**

**В некоторых проектах применяются стандарты, в которых предпринимается попытка ограничить размер каждого из модулей до некоторого приемлемого объема кода. На рис. 9.4 показано, как определить набор эвристических правил и представить его в форме дерева принятия решений, подтверждающего выводы Белфорда и других исследователей. Эти стандарты могут использоваться персоналом службы тестирования при разработке автоматизированных средств оценки всего проекта или отдельных программных модулей.**

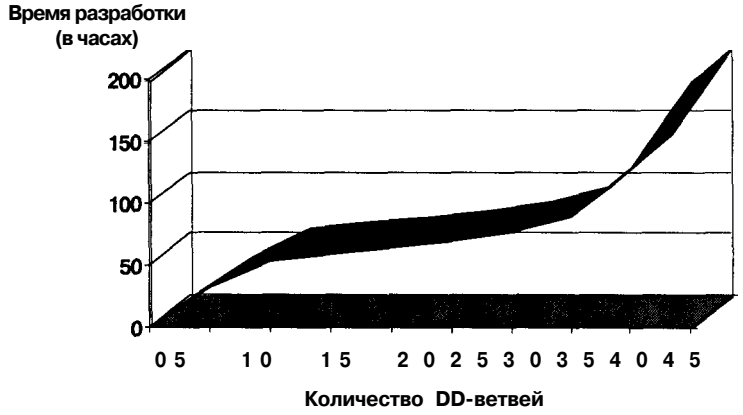
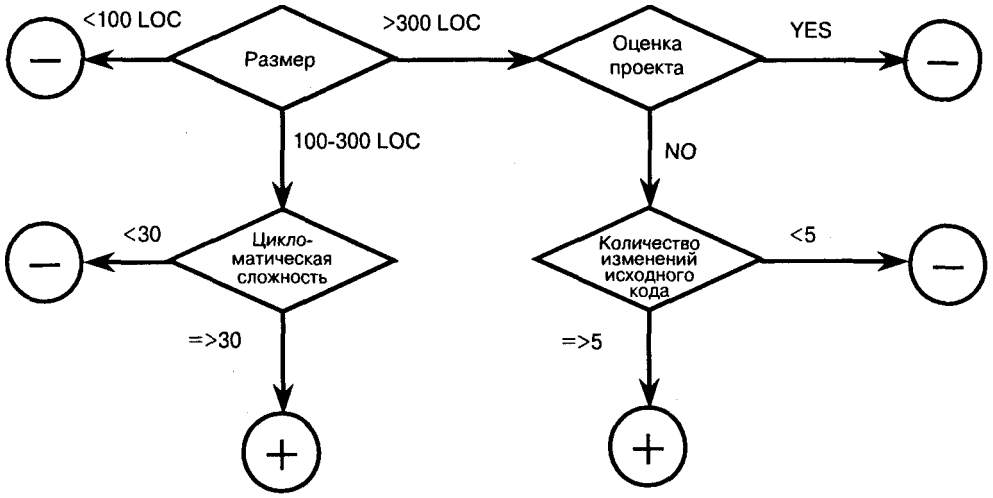



*Рис. 9.3. S-образная кривая Белфорда: с увеличением показателя цикломатической сложности более 30 затраты начинают возрастать по экспоненциальному закону.*




*Эвристическое правило:*
весьма вероятно, что модуль содержит много ошибок, если он имеет от 100 до 300 LOC, а его сложность составляет 15 или более

*Эвристическое правило:*
скорее всего, модуль содержит много ошибок, если он:
- имеет более 300 LOC
- оценка его проекта не проводилась
- изменялся более 5 раз

*Рис. 9.4. Определение стандартов, ограничивающих размер и сложность исходных модулей.*

# Формальная оценка

Имеет смысл периодически проводить формальную оценку каждого аспекта проекта разработки программного обеспечения. Может возникнуть вопрос, какое отношение периодические формальные оценки имеют к быстрому тестированию. Быстрое тестирование — это метод, в основе которого лежит выполнение каждой задачи на как можно более раннем этапе жизненного цикла и одновременное обнаружение и устранение любых недостатков, могущих возникать в процессе решения каждой задачи. Как правило, формальная оценка проводится накануне завершения какого-либо этапа, и объектами таких оценок становятся результаты, базы данных показателей проекта, календарные планы, перечни недостатков и тому подобные документы, полученные на каждом этапе жизненного цикла разработки.

Цель формальной оценки состоит в ознакомлении отдельных членов команды разработки с оценкой всех аспектов достигнутых результатов разработки с точки зрения руководства. Каждый разработчик должен увидеть и понять взаимозависимости между всеми полученными результатами, в особенности между теми, за получение которых он несет ответственность. Каждый разработчик должен увидеть и понять, какие показатели содержат информацию, имеющую отношение к выполняемой им работе. Специалисты по тестированию должны обратить особое внимание на элементы, которые должны оцениваться. Например, на состояние выполнения заказа на поставку испытательного оборудования или даты готовности новой документа-

ции, поскольку эта информация может влиять на план выполнения тестирования или разработку тестового случая. Иначе говоря, каждый участник выполнения формальной оценки оценивает результаты этапа с точки зрения их готовности к использованию. Хотя кое-кто может заявить, что эти обзоры служат для осуществления контроля над ходом разработки программного обеспечения со стороны руководства, мы смотрим на этот вопрос несколько иначе. Данные, полученные от членов команды, наряду со списками недостатков/нерешенных вопросов должны компилироваться и упорядочиваться руководителями различных уровней. Это позволит передать эти данные команде разработчиков, чтобы члены команды могли увеличить свою производительность или улучшить качество получаемых результатов.

Для программного обеспечения, разрабатываемого по контракту, в ходе каждой формальной оценки с участием клиента оценивается соблюдение сроков и условий контракта. Резюме этих оценок могут представляться клиент)', который должен быть уверен в корректности ведения разработки в рамках выделенного бюджета и оговоренных сроков. Ниже приведен перечень вопросов, который может подготовить и отслеживать руководитель разработки программного обеспечения или специалист по вопросам обеспечения качества, и который может быть представлен в ходе формальной оценки:

- Данные о размере программных продуктов или объеме изменений в программных продуктах отслеживаются, и в случае необходимости предпринимаются корректирующие действия.
- Размер программных продуктов или объем изменений в программных продуктах регулируются в соответствии с задокументированной процедурой.
- Данные по трудоемкости и затратам на разработку программы отслеживаются, и в случае необходимости предпринимаются корректирующие действия.
- Трудоемкость и затраты на разработку программы регулируются в соответствии с задокументированной процедурой.
- Критичные для выполнения проекта компьютерные ресурсы отслеживаются, и в случае необходимости предпринимаются корректирующие действия.
- Критичные для выполнения проекта компьютерные ресурсы управляются в соответствии с задокументированной процедурой.
- Календарный план разработки программного обеспечения проекта отслеживается, и в случае необходимости предпринимаются корректирующие действия.
- Календарное планирование и управление разработкой критичных связей и критичных ветвей программы выполняется в соответствии с задокументированной процедурой.
- Технические действия по разработке программного обеспечения отлеживаются, и в случае необходимости предпринимаются корректирующие действия.
- Отслеживаются вероятные просчеты в программном обеспечении, которые могут сказаться на затратах, ресурсах, календарном плане и технических аспектах.

- Вероятные просчеты в программном обеспечении, которые могут сказаться на затратах, ресурсах, календарном плане и технических аспектах, идентифицируются, оцениваются и документируются.
- Критичные взаимозависимости между разработками программного обеспечения идентифицируются, согласуются и отслеживаются в соответствии с задокументированной процедурой.
- Вероятные просчеты в программном обеспечении проекта идентифицируются, оцениваются, документируются и контролируются в соответствии с задокументированной процедурой.

В число присутствующих на формальной оценке могут входить представители компании/организации, выполняющей основную разработку, представители команды субподрядчика/партнера, команды организации-покупателя и команды потенциальных пользователей. Проведение формальной оценки позволяет руководству организации/проекта контролировать процесс разработки, в том числе просматривая отчеты и предпринимая корректирующие действия в отношении затрат, человеческих ресурсов, компьютерных ресурсов, технических проблем, критичных взаимозависимостей и вероятных просчетов. Секретарь должен записать действия, по которым достигнуты соглашения во время встречи, и по ее окончании перечень выбранных действий должен быть распространен среди исполнителей и отслеживаться вплоть до момента реализации всех действий. По окончании встречи перечень действий будет проработан руководителями нижнего уровня, способными устранить любые конфликты, которые могут возникнуть между группами. Независимо от того, является ли организация главным подрядчиком или субподрядчиком (в наше время в деловом языке чаще употребляется термин *"партнеры"),* важно проводить хорошо организованные совместные формальные оценки. Главное достоинство проведения хорошо организованных формальных оценок — это возможность согласовывать усилия и делиться приобретенным опытом.

Для команды тестирования формальные оценки ценны в первую очередь тем, что позволяют гарантировать применение всеми партнерами одних и тех же критериев прохождения/непрохождения тестов, создание во всех организациях-партнерах совместимых тестовых сред, разработку средств для немедленного распространения информации обо всех обнаруженных ошибках и просчетах, поддержание календарных планов последовательной поставки версий продукта с устраненными ошибками и достижение соглашений по условиям прекращения тестирования.

## Применение контрольных перечней

Поскольку управление разработкой программного обеспечения может оказаться очень сложным, большое распространение получила практика применения сборников извлеченных уроков и наиболее эффективных методик, оформленных в виде контрольных перечней. Контрольные перечни служат удобным напоминанием о действиях, которые должны выполняться в процессе жизненного цикла разработки. Обычно они документируются в форме утвердительно-вопросительных предложений типа "если... то... ?". Контрольные перечни не следует искать в Internet, выбирая те, которые кажутся подходящими. Более того, контрольные перечни не должны использоваться в качестве стандартов организации. Если контрольный перечень не

разработан на основе уроков, извлеченных в рамках данного проекта, польза от него будет невелика. В то же время, правильно составленные контрольные перечни служат мощным средством снижения вероятности просчетов в ходе разработки проекта.

При выполнении быстрого тестирования, в зависимости от выбранного жизненного цикла разработки, языка и средств, используемых для разработки, и множества других факторов, контрольные перечни, созданные для данного проекта, могут иметь различное назначение. Ниже приведены примеры пунктов контрольного перечня:

- Если требования включают в себя требования по доступности, то должен ли используемый в проекте текст подвергаться проверке на предмет соответствия стандарту компании, предусматривающему манипуляции доступностью?
- Если требования включают в себя требования по доступности, то отражен ли в плане тестирования факт применения средств тестирования доступности проекта на предмет соответствия стандарту?
- Если требуется применение процесса инспектирования на предмет соответствия стандартам компании, то прошли ли все участвующие в инспектировании 4-часовый курс обучения и сдали ли выпускной экзамен?
- Если температура является обязательным параметром, то обеспечивается ли возможность выбора пользователем отображения температуры по шкале Цельсия или Фаренгейта?
- Если "abc" — выбранное средство управления конфигурацией, которое будет использоваться в проекте, то заблокированы ли все версии исходных модулей, которые в настоящее время проверяются на предмет обновления?

Авторам доводилось встречать базы данных пунктов контрольных перечней, в которых утверждение отделялось от вопроса и которые содержали дополнительные поля, используемые в качестве ключей сортировки. Тем самым достигалась возможность изменения формы печати контрольных перечней в зависимости от текущего этапа и назначения перечней. Повторим еще раз: главное, чтобы при возникновении проблемы все члены команды сделали все возможное для создания пункта контрольного перечня, в котором было бы указано, что должно быть сделано данной командой для устранения данной проблемы, если она возникнет снова.

Как создаются пункты контрольного перечня? Процедура состоит в следующем: если причина жалобы клиента отражена в верхней части диаграммы Парето, то существует ли пункт контрольного перечня, в утвердительной части которого описана причина жалобы клиента, а в вопросительной — правило устранения основной причины неполадки? Да, для создания такого перечня требуется терпение и методичность, но выгода от его применения огромна.

# Аудит

Для того чтобы убедиться в соблюдении инструкций, процедур и стандартов организации, необходимо выполнить аудит в рамках организации. При отсутствии утвержденных инструкций, процедур и стандартов аудит лишен смысла и лишь создавал бы значительные неудобства. Уведомление, планирование, беседы и составление отчетов по результатам аудита — все это действия, имеющие решающее значение для успешного проведения аудита. Беседы должны планироваться так, чтобы изолировать

друг от друга следующие производственные уровни: руководство и функции поддержки (например, управление конфигурированием), обеспечение качества, клиенты/спонсоры/бизнес-аналитик, управление производством, администратор баз данных и другие непосредственные исполнители разработки/тестирования. В каждой беседе должны участвовать ведущий аудитор и второй аудитор. Ведущий аудитор ведет беседу, и оба аудитора фиксируют все сказанное в ходе нее. Между беседами аудиторы могут меняться ролями. Советы по выполнению каждого из основных действий приведены в таблице 9.1.

**Таблица 9.1. Советы по выполнению каждого из основных действий в ходе аудита**

| № п/п | Основные действия | Советы |
|---|---|---|
| 1 | Уведомление руководителей проекта | Необходимо по телефону согласовать удобное время проведения аудита и список кандидатов для участия в беседах в ходе аудита. Затем потребуется разослать сообщения электронной почты с уведомлением о сроках бесед с руководителями проекта, каждая из которых должна длиться около 1,5 часа. |
| | Уведомление сотрудников, выполняющих вспомогательные функции | Составьте сообщение электронной почты, адресованное 6-10 участникам, с приглашением принять участие в беседе длительностью 1,5 часа, прихватив с собой соответствующие материалы, в том числе переписку между ними и разработчиками, свои отчеты разработчикам и переписку с руководством. |
| | Уведомление других непосредственных участников разработки/тестирования | Составьте сообщение электронной почты, адресованное 6-10 участникам, с приглашением принять участие в беседе длительностью 1,5 часа, прихватив с собой соответствующие материалы, в том числе календарные планы, итоговую документацию и образцы отчетов руководителей проекта, сотрудников, выполняющих вспомогательные функции и собственные экспертные оценки. |
| | Порядок проведения бесед в ходе аудита | Следует запланировать не менее трех бесед, причем вначале должны проводиться беседы с сотрудниками, выполняющими вспомогательные функции, а конце — беседы с руководителями проекта. Остальные беседы должны проводиться в соответствии с графиком. |
| | Гарантия конфиденциальности | В ходе каждой беседы, после краткого вступления следует сделать следующее заявление: "Информация, предоставленная вами во время этой беседы, будет использована исключительно для улучшения процесса и качества продукта, но не для того, чтобы проверить вашу квалификацию или причинить какие-либо неприятности. Аудиторы будут делать пометки, которые будут использованы при составлении отчета, после чего они будут уничтожены. Наши отчеты будут представлены руководителям вашего проекта, но в них никак не будет отражено, кто и что сказал во время бесед. И безусловно, нигде не будет никаких ссылок на эти беседы в ходе аудита. Вся информация, задокументированная в итоговых отчетах, будет предназначена исключительно для постоянного совершенствования производственных процессов и качества продуктов. |

*Окончание табл. 9.1*

| № п/п | Основные действия | Советы |
|---|---|---|
| | Управление ходом встречи со стороны аудиторов | В ходе каждой беседы аудиторы должны поддерживать нужный ритм, профессиональный и производительный настрой и следить за тем, чтобы она проводилась в соответствии с графиком. Во время беседы недопустимы никакие персональные обвинения. Каждого участника следует призывать высказывать собственное мнение по каждой затронутой в ходе беседы теме. |
| | Выводы и подготовка отчета | По завершении каждых трех бесед оба аудитора должны работать вместе в течение 2-3 часов, чтобы "по горячим следам" систематизировать выполненные заметки в поисках улучшений, вытекающих, по меньшей мере, из двух бесед. Один из аудиторов должен составить черновик выводов и передать его для комментариев второму аудитору. Процесс должен продолжаться до тех пор, пока между аудиторами не будет достигнуто соглашение по формулировке и тону каждого вывода. |
| | Ознакомление с заключительными выводами | В некоторых случаях может планироваться ознакомление руководителей проекта с заключительными выводами с целью прояснения каждого вывода. Но доведение выводов до сведения членов группы разработки и вспомогательного персонала — задача руководителей данного проекта. |
| | Отчет с заключительными выводами и заметки, сделанные в ходе бесед | Аудиторы должны уничтожить все заметки, сделанные в ходе бесед. Они должны поместить отчет с заключительными выводами в архив аудиторских проверок и сделать запись о завершении аудита в журнале регистрации аудиторских проверок. |

**В** 1999 году нам довелось провести аудиторские проверки более 25 команд разработчиков, дабы удостовериться, что эти команды, занимавшиеся внесением изменений, решающих проблему 2000-го года, полностью выполнили соответствующее тестирование и аттестацию, и эта проблема должна быть решена к первому кварталу 2000 г. Результаты, полученные в ходе аудита проектов программного обеспечения, свидетельствовали о высоком профессионализме аудиторов во время проведения бесед и о точности сделанных выводов, которые указывали на любые возможные проблемы в ходе производственных процессов. Ценность проведения этой проверки для компании заключалась в том, что аудиторская проверка 10% команд разработчиков позволила в ходе выполнения проекта по решению проблемы 2000-ного года прийти к правильному заключению: изменения, внесенные в программное обеспечение производственного процесса в период между тестированием и аттестацией, не привели к созданию несовместимого программного обеспечения.

# Инспекции/критический анализ/ экспертные оценки

Экспертные оценки в той или иной форме используются в большинстве проектов по разработке программного обеспечения. Одна из задач персонала подразделений обеспечения качества состоит в выполнении статистических выборок, чтении и пометках пользовательской или клиентской документации. Эти действия квалифицируют как экспертную оценку. Иногда программист направляет листинги исходного кода только что созданного модуля и его проектную документацию другому программисту с просьбой прочесть код и снабдить его пометками. Это также называется экспертной оценкой. Программисты также делятся друг с другом отладочной информацией с целью устранения особо трудно обнаружимых ошибок. Участие в таком своеобразном клубе обмена информацией об ошибках тоже считается экспертной оценкой. Хотя эти неформальные действия позволяют исправить множество ошибок, существует множество ошибок, которые остаются незамеченными, в основном из-за недостаточной полноты и тщательности при проведении экспертной оценки. Специалисты по тестированию могут использовать экспертные оценки во время разработки сценариев тестирования, получения данных тестов и оценки результатов тестирования.

С целью совершенствования методов экспертной оценки в качестве более полных и тщательных средств поиска, документирования и исправления ошибок были разработаны методы формализованного критического анализа и инспекций. Критический анализ предполагает, что автор документа или исходного кода программы представляет коллегам свой подход к использованию данного компонента программного продукта. Группа коллег обсуждает предложенные автором подходы/алгоритмы, интерпретации требований или метод документирования, задавая автору вопросы или прося пояснений, пока вся группа не будет удовлетворена предложенными решениями. Тестировщики могут также использовать критический анализ во время планирования тестов, подготовки тестовых случаев, получения данных и оценки результатов тестирования.

Майкл Фаган впервые определил понятие инспекции во время работы в компании IBM [34]. Инспекции определяется в качестве одной из форм экспертной оценки и обладают рядом общих черт с неформальным просмотром кода и критической оценкой. Инспекции начинаются с представления автором листинга исходного кода и проектной документации модуля координатору инспекций.

# Распределение ролей и обязанностей в группе выполнения инспекций

Координатор инспекций выбирает четыре-пять инспекторов, один из которых назначается председателем комиссии по инспекциям. Координатор выбирает также время и место проведения инспекций. Он подготавливает для команды проведения' инспекций четыре или пять пакетов документации, включающих в себя копии листингов модуля и соответствующей проектной документации, несколько форм инспекций и соответствующих данному этапу контрольных перечней, составленных на

базе опыта предыдущих инспекций. Эти пакеты документации передаются членам комиссии не позже, чем за два дня до проведения заседания комиссии. Каждый член комиссии по инспекциям должен быть тщательно подготовлен. Председатель комиссии и ее члены должны знать, что они должны потратить от одного до двух часов на ознакомление с представленными на инспекцию материалами до начала заседания комиссии. Каждый из инспекторов может делать пометки на листингах исходного кода, а остальные комментарии вносятся в формы инспекций, входящие в состав пакета документации. Формы инспекций частично заполняются каждым членом комиссии до прихода на заседание. В том числе это относится к перечню возможных ошибок, областей, в которых возможны усовершенствования, замечаниям о несоблюдении соответствующих стандартов и т.п. Накануне заседания председатель комиссии должен предложить каждому члену комиссии еще раз убедиться в том, что они завершили просмотр материалов, и сообщить уточненное время и место проведения заседания. Календарный план типичной инспекции приведен в таблице 9.2, но при необходимости сроки выполнения отдельных этапов можно сократить до одного дня.

**Таблица 9.2. Календарный план проведения типичной инспекции**

| Этапы | Ответственный | Примечание |
|---|---|---|
| День 1 | Разработчик | Предоставляет координатору инспектируемые данные. |
| Дни 1-2 | Координатор | Разрабатывает и размножает пакет документации, назначает время/место заседания комиссии, приглашает 3-4 инспектора и председателя комиссии. |
| Дни 3-4 | Члены комиссии по освидетельствованию | Затрачивают 1-2 часа на прочтение/изучение представленных материалов, разрабатывая перечень замеченных ошибок. |
| Дни 4-5 | Члены комиссии по освидетельствованию | Проводят заседание комиссии. |
| День 5 | Председатель | Организует передачу данных от комиссии по инспекции, в том числе: ввод сведений об обнаруженных ошибках в базу данных недостатков, ввод полученных в результате инспекции показателей в базу данных показателей проекта и ввод результатов анализа основных причин возникновения новых типов ошибок в базу данных контрольных перечней устранения ошибок, сгруппированных по этапам. |
| День 5 | Координатор | Принимает от председателя комиссии пакеты документов инспекции и отчеты, анализирует перебои в процессе инспекции, если таковые имеют место, и принимает меры для совершенствования процессов/баз данных инспекций. |

В ходе заседания председатель комиссии рассаживает участников заседания в соответствии с рис. 9.5. Решение о количестве разработчиков и специалистов по тестированию в составе комиссии принимается координатором инспекций и зависит от характера пакета представляемой на инспекцию документации. Например, если это план тестирования, в комиссии должны участвовать один-два специалиста по тестированию и ни одного или один разработчик программного обеспечения. В то же время, если представленный на инспекцию пакет содержит исходный код на языке C++,

диаграмму потоков данных и диаграмму отношений объекта, за столом комиссии могут присутствовать два разработчика программного обеспечения и ни одного или один тестировщик. И, наконец, если пакет содержит список всех компонентов данной версии продукта, то более подходящим может быть состав комиссии, представленный в правой части рис. 9.5, когда место разработчика программного обеспечения или специалиста по тестированию занимает руководитель разработки версии или лицо, отвечающее за управление конфигурацией программного обеспечения.

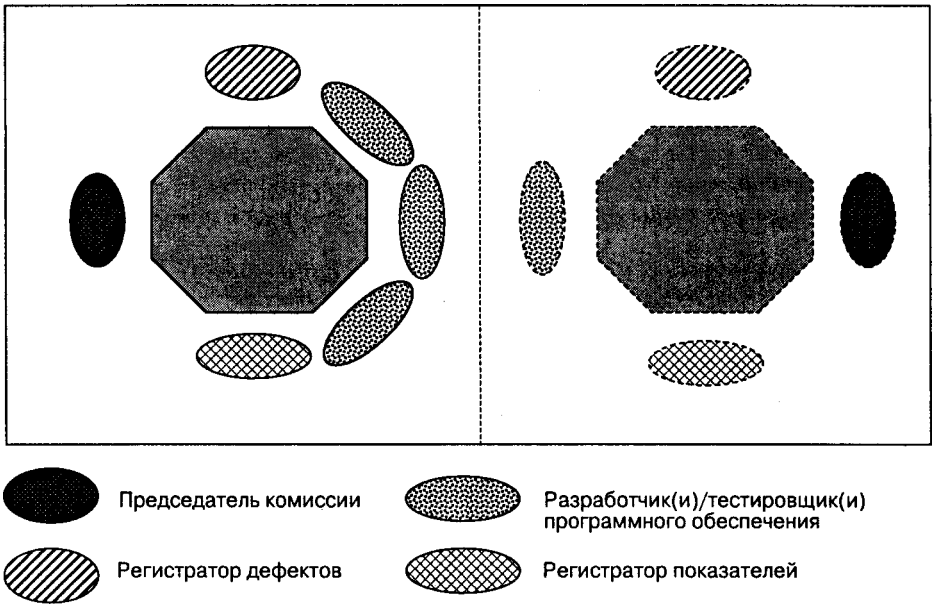



*Рис. 9.5. Различные расположения мест за столом комиссии по инспекции.*

Задача председателя комиссии по инспекции состоит в выполнении функций, способствующих проведению заседания комиссии. В этом качестве председатель комиссии обеспечивает, чтобы снабженный документацией и развернутый в соответствии с ранее упоминавшейся подготовкой процесс инспекций был выполнен в полном объеме. Задача регистратора недостатков заключается в фиксации всех обнаруженных комиссией дефектов, удаление любых дублированных замечаний, руководство процессом уточнения формулировок каждого из обнаруженных просчетов, классифицирование ошибок по уровню серьезности и присвоение им номера SEV перед тем, как они будут внесены в формы инспекции и представлены координатору. Задачи регистратора показателей сводятся к обеспечению учета дефектов (ошибок) и возможных основных причин их возникновения или этапов жизненного цикла, на которых они, скорее всего, были допущены. Кроме того, в задачи входит и фиксация любых других текущих показателей инспекции проекта, таких как: общее время, затраченное членами комиссии на подготовку к инспекции, количество инспекторов в комиссии, общее время, затраченное на инспекции, общее количество ошибок, сведения о которых были введены в базу данных недостатков, и т.п. Эти данные вводятся в формы инспекций. Председатель собирает формы у всех комиссии, после чего

обновляет базы данных дефектов, показателей и контрольных перечней и представляет печатную копию сводного пакета документации координатору инспекций.

## Отчетность о процессе выполнения инспекций

Координатор инспекций обеспечивает максимально быстрый ввод сведений об ошибках, зафиксированных в пакете документации по инспекции, в общую базу данных дефектов проекта. Еще одна отличительная черта быстрого тестирования заключается в том, что некоторые из этих недостатков помечаются как "блокирующие", т.е. данная ошибка непосредственно препятствует дальнейшему тестированию конкретной функции. О выявлении блокирующих ошибок и ошибок уровня SEV 1 (катастрофических) следует немедленно сообщать команде разработки, чтобы она смогла выполнить отладку.

Основная цель быстрого тестирования состоит в обнаружении и исправлении ошибок в кратчайшие строки. Сохранение данных о каждой ошибке может существенно повысить точность и значимость процесса инспекции. Контрольный перечень, составленный на основе полученного опыта, должен создаваться для каждого этапа жизненного цикла. Эти сгруппированные по этапам контрольные перечни должны стать действующими документами, отражающими данные о новых ошибках, которые получаются в результате какой-либо инспекции. Комиссия по инспекции может выполнить общий анализ каждой ошибки с целью выяснения этапа, на котором она была допущена. Эта комиссия должна добавить полученные сведения в контрольный перечень данного этапа, чтобы следующая комиссия, выполняющая инспекцию материалов в рамках этапа, наверняка обнаружила ошибку, аналогичную обнаруженной ранее. Такая обратная связь, реализуемая для каждой ошибки, делает контрольный перечень инспекций постоянно совершенствующимся документом. А это, в свою очередь, является отличительной чертой организации, которая обучается на собственных ошибках.

## Показатели процесса инспекции

Показатели, полученные в результате проведения инспекций, непосредственно загружаются в модель ошибок проекта. Количество дефектов на тысячу исходных строк кода объединяется с другими показателями инспекций, получаемыми в течение определенного периода времени, например, ежедневно, еженедельно или в течение этапа разработки, и добавляется к входным данным программы оценки погрешности программного обеспечения (Software Error Estimation Program, SWEEP). Программа SWEEP подробно описывается в главе 11.

## Использование электронной почты или другого электронного приложения для ускорения процесса инспекции

Для обеспечения быстрого тестирования на протяжении всего жизненного цикла проекта разработки программного обеспечения должны интенсивно использоваться разнообразные способы повышения эффективности. Существует несколько техноло-

гий, которые применяются не только к процессу инспекции, как описано в этом разделе, но и к другим процессам в рамках цикла разработки. Предполагается, что команда разработки программного обеспечения создает исходный код на компьютерах, которые связаны между собой локальными сетями (local area networks — LANs). За счет использования серверов локальной сети или архитектуры типа сервер-хранилище данных (storage area network, SAN) любой участник проекта должен иметь возможность доступа к общим хранилищам данных проекта. В таких сетях поддерживаются структуры каталогов, которые должным образом организуют данные по разработке и тестированию. При этом предусматривается их регулярное резервное копирование в соответствии с установленным календарным планом. В качестве альтернативы технологии LAN/SAN управление хранилищем данных проекта может осуществляться также с помощью некоторого электронного приложения, действующего на Web-сайте проекта в рамках корпоративной сети организации. Это приложение должно заботиться о сохранении документации по разработке в системе управления конфигурациями, которая реализует строгий контроль версий и регулярно, в соответствии с календарным планом, выполняет резервное копирование.

Используя встроенный календарь и обмен сообщениями по электронной почте локальной сети организации, персонал, занятый проектированием, может резервировать конференц-залы и приглашать участников каждой инспекции. Во время разработки проекта, при которой задействованы методы быстрого тестирования, подобное повышение эффективности ведет к уменьшению временных затрат координатора инспекций на резервирование конференц-зала и приглашение всех инспекторов на время, не связанное с какими-либо накладками. Соблюдение этого условия обеспечивается координатором, который при выборе подходящего времени и места заседания может просматривать календарные планы всех кандидатов в инспекторы. Корпоративная инфраструктура, образующая корпоративную сеть организации, предоставляет также доступ к пакету документации по инспекциям, который может быть оформлен в виде набора гиперссылок в сообщениях электронной почты на доступные только для чтения версии форм и документов в общих хранилищах данных. Кроме того, копии одних и тех же общих файлов могут добавляться в виде присоединений к календарным назначениям, рассылаемым всем инспекторам.

Инспекция — это инструмент статического тестирования, который может применяться для всех программных документов, начиная с этапа разработки требований и заканчивая этапом приемочных испытаний. Быстрое тестирование предполагает постоянный контроль на предмет возникновения ошибок, которые должны быть обнаружены и исправлены как можно скорее после их возникновения. Поскольку инспекции должны выполняться применительно к документу, который разрабатывается на данном этапе жизненного цикла, все обнаруженные в ходе инспекций дефекты должны отслеживаться вплоть до основной причины их возникновения. Затем комиссия по инспекциям должна обновить контрольные перечни данного этапа, чтобы будущие комиссии не пропустили эти же или аналогичные ошибки. Ниже приведены характеристики инспекций, которые соответствуют требованиям быстрого тестирования:

- В результате инспекций создаются списки ошибок, каждой из которых присвоен номер уровня серьезности, характеризующий приоритет процесса ее устра-

нения. Список ошибок вносится в базу данных дефектов и используется при внесении изменений в систему, положенную в основу будущих версий.

- В результате инспекций создаются версии контрольных перечней для одного или более этапов, на основе которых инспекторы выполняют анализ основных причин возникновения дефектов. Эти контрольные перечни полностью готовы к использованию во время следующего выполнения данного этапа в рамках текущего или другого проекта.
- В результате инспекций определяются показатели, которые управляют характеристической моделью ошибок проекта и служат входными данными для программы оценки погрешности программного обеспечения (SWEEP), используемой для прогнозирования скрытых дефектов, остающихся в самом программном продукте. Эти показатели служат также основой для постоянного совершенствования процесса выполнения инспекций. Программа SWEEP подробно описывается в главе 11.
- Инспекции способствуют повышению квалификации инспекторов, которые, возвращаясь к выполнению своих обязанностей разработчика, имеют более глубокие знания о продукте или требованиях, архитектуре, проекте и запрограммированных алгоритмах, а также о типах ошибок, которые следует отслеживать во время работы над их собственными рабочими продуктами.

Из этого определения процесса инспекции, которое было разработано специально для быстрого тестирования, должно быть понятно, как именно объединять эту форму статического тестирования с реальным процессом разработки, и какое значение придается быстрому повышению качества результирующего продукта. Процесс инспекции, выполняемый в ходе быстрого тестирования, сокращает календарные сроки за счет исправления ошибок в ходе разработки, а не ближе к ее завершению.

## Формальная верификация

Верификация — это действия по проверке выполнения на и-ом этапе разработки того, что было запланировано на (п -1)-ом этапе. Во время создания архитектуры системы и при определении требований производительности специалист по тестированию должен сравнивать обусловленные архитектурой ресурсы производительности с общими требованиями. Во время тестирования рабочей проектной документации тестировщик будет сверять эту документацию с документацией по архитектуре, которая была разработана на этапе высокоуровневого проектирования. На этапе создания кодов тестировщик будет проверять код, сравнивая его с рабочей проектной документацией. Например, в соответствии с рабочей проектной документацией для реализации некоторой функции должны быть созданы пять модулей, причем одним из них является комплекснозначный алгоритм быстрого преобразования Фурье (БПФ). В такой ситуации среди множества тестовых случаев должен существовать случай, ориентированный на проверку кода БПФ во время выполнения и обеспечивающий оценку точности, эффективности и всех встроенных функций, которые могут присутствовать.

## Языки на основе спецификаций

Одно из средств в инструментальном наборе специалиста по тестированию носит название программного доказательства. Данное средство относится к категории формальной верификации, поскольку при этом подтверждается соответствие кода его рабочему проекту. Программное доказательство начинается с формулировки на специальном языке теоремы или леммы (называемой спецификациями) непосредственно в основном коде, а именно — в заголовках каждого главного блока кода. Затем пользователь интерактивно взаимодействует с автоматизированным средством доказательства теоремы с целью определения соответствия кода приведенной теореме или лемме. Такого рода языки называют языками на основе спецификаций или языками утверждений, а на инструментальное средство ссылаются как на средство формальной верификации. Хотя в настоящее время подобные средства редко применяются при промышленной разработке программного обеспечения, тем не менее, они заслуживают того, чтобы включить их в арсенал инструментов тестирования.

## Автоматизированное доказательство теорем

У нас имеется определенный опыт выполнения программных доказательств с использованием компьютеризованной утилиты доказательства теорем Gypsy [18] для проверки приблизительно тысячи строк кода. Принцип работы утилиты Gypsy заключается в следующем. Сначала тестировщик помещает формулировки теорем и лемм вместе с блоками кода, написанного на языке Gypsy (один из вариантов языка Pascal). Затем встроенный в Gypsy модуль доказательства теорем проверяет правильность реализации блоком кода всех утверждений, сформулированных в теоремах и леммах. Алгоритм, корректность работы которого довелось доказывать с помощью этого средства формальной проверки, носил название Collision Avoidance (исключение столкновений). В настоящее время упомянутый алгоритм реализован в компьютерных системах Федерального авиационного агентства США для предупреждения самолетов о грозящих столкновениях. Для достижения этой цели на самолетах, а также на высоких зданиях и горах устанавливаются радиомаяки-ответчики. Эти маяки-ответчики передают цифровые пакеты, содержащие данные о таких динамических характеристиках полета, как высота, скорость и направление полета. На базе математических моделей динамики полета выполняется имитационное моделирование для данных из полученных цифровых пакетов, цель которого заключается в определении вероятности столкновения, а также в выработке необходимой тактики предотвращения столкновения. Эти данные автоматически и заблаговременно передаются на борт самолетов с пересекающимися траекториями полета и позволяют предотвратить столкновение. Таким образом, очевидно, что код алгоритма исключения столкновений должен быть корректным, причем доказуемо корректным.

Вначале программа исключения столкновений была переведена на язык Gypsy, **и** для каждого блока кода были сформулированы теоремы. Затем при помощи возможностей Gypsy по доказательству теорем было установлено, что все теоремы в коде реализованы. В ходе процесса была обнаружена одна программная ошибка, после чего в исходный код были внесены соответствующие исправления. Процесс оказался безболезненным и действительно эффективным.

## Средства автоматизации тестирования

Существует множество средств автоматизированного статического тестирования. Большинство из них являются специализированными средствами, а не одним универсальным инструментом, которое могло бы применяться во всех случаях. Еще одна отличительная особенность этих средств статического тестирования — их зависимость от языка. Обнаружение ошибок в исходном языке или в языке документации предполагает выделение ошибок, в том числе орфографических, синтаксических, ошибок, связанных с неопределенными переменными и указателями, незакрытыми программными конструкциями, неопределенными ссылками, включая Internet-адреса, и т.д.

## Прослеживаемость требований

Прослеживаемость требований означает сохранение формулировки требования от момента его указания до момента выполнения. Из главы 8 уже должно быть известно, что результатом совместной разработки требований к приложению (Joint Application Requirement, JAR) является набор требований к программному обеспечению. Последующие требования разбивают исходные на набор производных требований, которые, как правило, документируются в спецификации проекта программного обеспечения (Software Design Specification, SDS). Мы уже рассмотрели существующие возможности статического тестирования этой переформулированной версии требований, которые позволяют обнаружить пропущенные, неверно сформулированные, двусмысленные и попарно конфликтующие требования, приводящие к ошибкам в спецификации проекта программного обеспечения. Обнаружение ошибок в требованиях — наибольший выигрыш, даваемый всеми формами тестирования. В рамках парадигмы быстрого тестирования определяется матрица прослеживаемости требований (Requirements Traceability Matrix, RTM), за счет использования которой группа разработки, включая тестировщиков, демонстрирует свою заинтересованность в соблюдении и полноте тестирования требований. При этом на протяжении всего жизненного цикла разработки применяются как статические, так и динамические средства тестирования.

С другой стороны, если применяется методика, отличная от быстрого тестирования, и тестирование начинается лишь тогда, когда разработка программного обеспечения подходит к концу, то для получения RTM, которая будет использоваться для планирования тестирования в ходе оставшейся части жизненного цикла разработки, придется выполнить реконструирование требований.

## Программа контроля единиц измерения физических величин

Огромную помощь в разработке научных программ оказывает программа контроля единиц измерения физических величин. Эта программа представляет собой статический препроцессор исходного кода, который проверяет, что результат каждого математического выражения выражается в соответствующих единицах измерения физических величин. Рассмотрим выражение для определения частоты (R) или скоро-

сти: *Rl = D1/T1,* где *D1* — пройденное расстояние, а 77 — время передвижения. Это выражение справедливо только в том случае, если единицы измерения физических величин совместимы. Иначе оно будет неверным. Например, если расстояние выражается в километрах, а время — в минутах, то скорость, вычисляемая в соответствии с этим выражением, должна выражаться в км/мин. Если это значение скорости необходимо вставить в другое выражение, в котором скорость должна выражаться в милях в час (миль/ч), вначале потребуется выполнить преобразование *R1* (км/мин) в *R2*(миль/ч).

При реализации в качестве статического средства контроля программа контроля единиц измерения физических величин содержит дополнительные спецификации ввода всех переменных, хранящих физические величины, которые определены в специально сформатированных строках исходного кода. При каждом использовании или определении этих переменных формулы проверяются на предмет того, что переменные в левой части выражения (результаты) получают единицы измерения на основе вычислений и на базе объявленных единиц измерения, которые связаны с переменными из правой части (входные переменные).

## Символьное выполнение

Символьное выполнение — это технология подстановки символов в формулы, закодированные внутри языка программирования с выполнением алгебраического объединения символов в соответствии с алгоритмом. Рассмотрим код функции вычисления синуса SINE, приведенный на рис. 9.6. Этот алгоритм содержит минимальное количество произведений, как показано в следующем выражении:

$$\sin(x) = x*(1 - x^2*(1/3! - x^2*(1/5! - x^2*(1/7! - x^2*(\ldots)))))).$$

```
      REAL FUNCTION SINE  (P,EPS)
      ERROR = P
      SUM = P
      DO J = 3,1000,2
              ERROR = ERROR *  (P**2)  /  (J*  (J+1))
              SUM = SUM -  ((J+1)/2)  * ERROR
              IF  (ABS(ERROR)  .LT. EPS)
              THEN
              GO  TO  30
      ENDIF
      ENDDO
30    SINE = SUM
      RETURN
      END
```

*Рис. 9.6. Функция SINE, реализованная с использованием разложения вряд Тейлора*

Программа символьного выполнения осуществляет математические упрощения для реализации приведенных в таблице 9.3 шагов алгоритма, который представляет собой известный метод разложения в ряд Тейлора.

Полученные результаты можно сравнить со значением функции sin(x), полученным из большинства математических справочников, а именно:

$\sin(x) = x - x3/3! + x5/5! + x^7/7!...$

Повторимся еще раз: из этого описания понятно, почему символьное выполнение является средством статического тестирования. Это связано с тем, что для выполнения алгоритма не требуется ввод числовых данных.

**Таблица 9.3. Символьное выполнение алгоритма, приведенного на рис. 9.6**

| J | Ошибка | Сумма |
|---|---|---|
| 3 | $P*(P^2/12) = P^3/12$ | $P-(4/2)*(P^3/12) = P - P^3/6$ |
| 5 | $(P^3/12)*P^2/30-P^5/360$ | $P-P^3/6+(6/2)*P^5/360 = P-P^3/3!+P^5/120$ |
| 7 | $P^5/360*P^2/56 = P^7/20160$ | $P-P^s/6+P^5/120-4*(P^7/20160) = P-P^3/3!+P^5/5!-P^7/7!$ |

J $P^J/(J!*(J+1)/2)$ для j = 3,n,2 P-SUM(pj/J!)для J=3,n,2

# Листинги перекрестных ссылок

Листинги перекрестных ссылок — это зависящее от языка средство тестирования и отладки, которое индексирует использование переменных внутри каждого модуля. Некоторые перекрестные ссылки включают в себя список строк исходного кода, в которых переменные используются или определяются. Это позволяет тестировщику выявлять неправильно введенные имена переменных, приводящие к возникновению программных ошибок. Неправильно введенная переменная будет либо неопределенной, либо использоваться в вычислении, в котором присутствовать не должна. В большинстве компиляторов такая функция реализована.

# Программы улучшенной печати

Программы улучшенной печати — это зависимое от структурированного языка средство тестирования, обеспечивающее корректное завершение каждой структурной конструкции. Например, если в конструкции IF-THEN-ELSE-ENDIF отсутствует ключевое слово ENDIF, при улучшенной печати структура будет выглядеть неправильно выровненной. Это часто приводит к ошибке, поскольку часть ELSE может содержать выражение, которое не должно выполняться. Во многих компиляторах структурированных языков такая функция реализована.

## Средства сравнения версий

Средства управления конфигурациями обеспечивают выбор в качестве последней версии такой, которую можно проверить на предмет изменений. Эти средства служат для упрощения работы создателей программного обеспечения. Например, если два разработчика проверяют одну и ту же версию исходного кода и начинают вносить в нее изменения, то когда они одновременно делают попытку вернуться обратно к проверке, возникает дилемма. Определить, какая версия должна проверяться первой, может помочь только какой-нибудь механизм блокировки, предоставляемый средством управления конфигурациями. После этого оба автора могут прийти к соглашению о способе объединения конфликтующих изменений. Еще одно применение средства сравнения версий связано с проверкой на полное совпадение файла и одной из версий, поддерживаемых системой управления конфигурациями.

Ряд дополнительных функций средства управления конфигурациями помогают испытателям определить, вносились ли изменения в исполняемый код или в файлы данных между двумя последовательными версиями. Эти функции называются *средствами сравнения версий*. Средство сравнения версий осуществляет сравнение одной ячейки за другой, выделяя различия между двумя ячейками. При этом термин "ячейка" может означать строку исходного кода или поле данных внутри записи. Специалисты по тестированию и разработчики должны обмениваться информацией об областях изменений в новых версиях. Как правило, это делается при помощи записей. Это же средство может использоваться и в качестве инструмента предотвращения ошибок во время проверки измененных областей. Естественно, неизменные области могут потребовать регрессивного тестирования для обнаружения возможных побочных эффектов, но тестовые случаи, предназначенные для проверки неизмененных областей кода, наверняка не потребуют повторного выполнения. Таким образом, использование средства сравнения версий может привести к повышению эффективности работы персонала, занятого быстрым тестированием.

## Тестирование алгоритмов

Если перед тестировщиком поставлена задача тестирования модуля, который реализует преобразование данных по определенному алгоритму, наиболее строгий подход предполагает разработку входных и ожидаемых результирующих данных, которые позволят испытать возможности, варианты выполнения и условия возникновения ошибок алгоритма. Например, если перед специалистом по тестированию стоит задача проверки алгоритма быстрого преобразования Фурье, проверке должны подвергаться несколько свойств этого преобразования. Поскольку быстрое преобразование Фурье является линейным, тестировщик создает тестовый случай для проверки равенства $T(a*f, + b*f_2) = a*T(f,) + b*T(f_2)$. Поскольку быстрое преобразование Фурье является обратимым, тестировщик может выполнить прямое преобразование отдельных частот в спектральную функцию, а затем подвергнуть эту спектральную функцию обратному преобразованию и убедиться в совпадении результата с исходными отдельными частотами, т.е. в том, что $T~'[T\f)]=f$. На этом этапе тестировщик может прийти к выводу, что код реализует обратимое линейное преобразование, но вопрос о том, является ли алгоритм действительно преобразованием Фурье, остается

открытым. Специалисту по тестированию наверняка придется создать данные для фиксированной частоты, для которой спектральная функция имеет только одну ненулевую составляющую на выбранной частоте, и убедиться, что ее амплитуда имеет правильное значение. За этим тестовым случаем могут прогоняться случаи для следующей "чистой" частоты и так далее, пока не будут протестированы все частоты и амплитуды.

Однако этот строгий подход относится к категории динамического тестирования, которое представляет собой тему главы 10. Значительно больший объем тестирования этого алгоритма можно предпринять во время его разработки с одновременным использованием статического тестирования. При разработке требований упомянутый алгоритм формулируется, например, так: "Пользователь должен располагать утилитой, которая будет выполнять быстрое преобразование Фурье для комплексных данных, длина которых составляет степень двух, вплоть до $2^{10}$ включительно". Эта формулировка понятна персоналу инженерной или физической лаборатории, но разработчик программного обеспечения может не обладать достаточной математической подготовкой. Во время эскизного проектирования это требование должно быть разбито на более понятные части. Прежде всего, входные и выходные данные помещаются в два двумерных массива длиной я, имеющие тип COMPLEX и максимальную длину 210 во флаг ошибки типа INTEGER и в переменную Nтипа INTEGER, значение которой лежит в интервале [2,10]. Приведенная информация эскизного проекта определяет пользовательский интерфейс, который дает возможность подготовить прототип, или заглушку, для помещения ее в графический интерфейс пользователя на этапе эскизного проектирования до того, как будут получены необходимые разъяснения от персонала инженерной или физической лаборатории.

На этапе рабочего проектирования алгоритм быстрого преобразования Фурье должен быть тщательно проанализирован. Здесь же понадобиться выбрать либо его оригинальную версию, разработанную Кули (Cooley) и Туки (Tukey) в 1965 году [12], либо какую-нибудь другую модификацию из более новых. Потребуется принять ряд решений относительно порядка обработки. Например, нужно ли применять обратный порядок следования разрядов выходных данных перед завершением преобразования, или предварительно сортировать таблицу данных синусов и косинусов с использованием алгоритма с обратным порядком следования разрядов, после чего сохранять данные в фиксированной области памяти с тем же обратным порядком следования разрядов. Необходимо также принять решения относительно требования к точности алгоритма. Это решение должно подкрепляться задокументированными ссылками на техническую литературу, которая обосновывает требования по тестированию точности результирующего алгоритма преобразования. Кроме того, нужно определить и требования ко времени выполнения преобразования, чтобы можно было проанализировать, достаточно ли будет скалярного процессора, или же потребуется процессор, в архитектуре которого реализованы векторные инструкции и другие функции обработки массивов.

Этапы проектирования программного обеспечения служат фундаментом для разработки требований, и все тестирование, выполняемое во время проектирования, за исключением динамического тестирования прототипов, является статическим. Многие решения, которые должны быть приняты во время рабочего проектирования, могут оказаться ошибочными и вести к значительным перерасходам средств или к

формулированию требований, которые не могут быть выполнены. Подход к этому вопросу с использованием быстрого тестирования заключается в сосредоточении усилий на поиске компромиссных решений на этапе рабочего проектирования. Затем компромиссные решения могут использоваться в качестве средства предотвращения изменчивости требований - одной из основных причин повышения затрат в ходе разработки с традиционным каскадным жизненным циклом. Компромиссные решения - это форма оптимизации перед утверждением окончательного варианта проекта, которая предотвращает многократное выполнение дорогостоящих разработок и динамического тестирования для отброшенных проектов/реализаций. В большинстве финансовых оценок Института технологий программного обеспечения (Software Engineering Institute, SEI) финансовые инспектора фиксируют многочисленные нарушения в области проектирования программного обеспечения. Мы называем это явление "выбоинами" процесса разработки - "ямами" на дороге, которые иногда могут повредить движущийся по ней проект. Быстрое тестирование - это искусство расстановки по всему жизненному циклу разработки специалистов по тестированию, которые обнаруживают и сообщают о недостатках, как только они появляются.

### ПРИМЕР: АНАЛИЗ ОБЩЕЙ ОШИБКИ ОКРУГЛЕНИЯ (ГЭРИ КОББ)

Еще одна область применения статического тестирования, которая особенно характерна для сферы научного программирования, — это анализ общей ошибки округления. Для того чтобы убедить в необходимости выполнения этой формы тестирования, стоит лишь' сослаться на один реальный случай из личной практики. В начале семидесятых компания, в которой я работал, заключила с одним из клиентов контракт на перенос их программного обеспеч. . . . .з скалярного компьютера в векторный. В части контракта, касающей ся приемки конечных продуктов, было оговорено следующее условие: на векторном. компьютере преобразованные программы будут выполняться в 256 раз быстрее и вьщавать те же шестнадцатеричные результаты, что и при выполнении на скалярном компьютере.

К моменту назначения меня на должность руководителя группы преобразования программного обеспечения, я выполнил оценку основных характеристик каждой из основе ных преобразуемых систем, как с точки зрения времени выполнения, так и в плане по-' лучения распечаток входных и выходных переменных при прогоне программ для данных, которые владельцы приложений считали типовыми. Мне казалось, что это должно было помочь выполнить условия получения приемлемой производительности для выбранного, метода преобразования. Вскоре после начала преобразования программного обеспечения один из моих сотрудников пришел к шутливому выводу: одно из выходных шестнадцатеричных значении было названо WWM, что означало "worldwide mass" ("масса земного шара ). Это значение было результатом суммирования масс всех ячеек сетки' наброшенной на весь земной шар. После совместной работы мы установили, что такое вычисление приводило к переполнению значения с плавающей точкой при обходе поимерно половины земного шара, поскольку накапливаемая сумма становилась настолько: большой, что добавление массы остальных ячеек сети не вело к изменению суммы Более того, скалярный компьютер выполнял 36-разрядные операции с плавающей точкой то время как векторный компьютер мог выполнять 32- или 64-разрядные векторные операции с плавающей точкой. Мы явно оказались в проигрышной позиции. Если выпопнять сложение с одинарной точностью, переполнение возникло бы прежде, чем удалось добраться даже до середины сетки (т.е. шестнадцатеричный результат отличается от получаемого на скалярном компьютере). Если же выполнять суммирование с двойной' точностью, вычисление выполнялось бы медленнее (невозможно было получить 256-кратное увеличение производительности), и переполнение начинало бы возникать при обходе примерно 3/4 сетки (шестнадцатеричный результат отличается от первоначаль-

**Для того чтобы отстоять примененный нами метод достижения оговоренной в контракте производительности в данном случае, пришлось доказать, что базовая программа получает неверное значение массы земного шара и поэтому преобразованная программа не должна получать такой же ошибочный результат.**

**Приложение анализа общей ошибки округления суть технология статического тестирования, берущая свое начало из области численного анализа, который подробно преподается на университетских курсах по теории вычислительных машин и систем. 8 этой теории каждая переменная входных данных имеет связанный с ней вектор ошибок. Если X,=C,+c„ а y,—D,+d„ то ошибка E определяется следующим выражением:**

**SUM $(X_i + Y_i)$=SUM$(C_i + D_i)$ + $E(N)$ для $i=1, \ldots, N$, где E(N)=SUM(c,+ d,) для i=1....N**

**Далее, поскольку при считывании входных значений компьютер всегда усекает действительные значения *X,* и *У„* значения c, и d, всегда будут неотрицательными усеченными значениями. Следовательно, функция ошибок *E(N)* монотонно возрастает и является неограниченной сверху. Этот численный анализ может быть выполнен применительно к более сложным алгоритмам для проверки их ограничений по стабильности.**

**Подводя итог этому реальному примеру, отметим следующее: тестировщики, имеющие дело с научными приложениями, должны уметь выполнять численный анализ сложных алгоритмов, чтобы определить граничные условия компьютерных имитаций, в которых задействованы эти сложные алгоритмы. Хотя упомянутые требования могут казаться непомерно высокими для тестировщиком, они помогают показать ценность специалистов, обладающих подготовкой в области численного анализа или общей математической подготовкой. Такие специалисты по тестированию смогут обнаружить нестабильные алгоритмы и отыскать для них стабильные альтернативы, которые окажутся гораздо более предпочтительными для компьютерных имитаций, особенно в сфере научного программирования.**

# Диспетчер тестирования

Роль диспетчера тестирования заключается в организации и направлении повседневного продвижения и отслеживании состояния тестирования с использованием подхода быстрого тестирования. В небольших проектах эту роль может выполнять руководитель проекта. В средних проектах эта должность может называться "руководитель тестирования". В случае очень больших проектов, во время динамического тестирования эти обязанности могут выполняться отдельным, полностью занятым только этой работой лицом, работающим с командой тестирования, специалистами по сетям, администраторами баз данных, техниками лаборатории тестирования и бизнес-аналитиками, которые представляют интересы сообщества пользователей. При выполнении тестирования применимости эта должность может называться заведующим лабораторией применимости. Такой заведующий призывает посетителей из сторонних организаций испытать новый набор продуктов и технологий на предмет удобства его использования в реальных сценариях.

Роль диспетчера тестирования заключается в изучении планов тестирования проекта и реализации следующих координационных действий:

- Определение и управление календарными графиками загрузки персонала, которые потребуются для прогона полного набора тестовых случаев.
- Обеспечение соответствия порядка выполнения тестов потоку данных в программном обеспечении.
- Документирование сценариев тестирования.

- Проектирование форм отчетности по тестированию, которые будут заполняться, анализироваться и сравниваться с данными результатов прогона тестов.
- Сбор данных о состоянии тестов, выполняемых в реальном времени, и составление соответствующих отчетов.

Как видите, роль диспетчера тестирования в чем-то аналогична роли режиссера кинофильма или мультимедиа-продукта. Без диспетчера тестирования отдельные тестировщики, каждый из которых работает с собственным поднабором тестов, создавали бы хаос из-за недостаточной согласованности календарного графика выполнения тестирования, тестовых платформ, недостаточно полного совместного использования файлов, отсутствия "снимков" результатов тестирования и резервных копий, которые потребуются для генерирования среды каждого тестового случая. Параллельное выполнение тестирования несколькими специалистами с использованием независимых и разделяемых сетевых ресурсов требует наличия организованного плана, согласованного выполнения и обобщения результатов. Защита интересов конечных пользователей, выполняется ли она бизнес-аналитиками, сопровождающими большие действующие информационные системы, или псевдо-конечными пользователями коммерческого продукта, требует присутствия специально назначенного лица, а именно диспетчера тестирования. Диспетчер должен обеспечивать готовность календарных графиков, ресурсов и данных к моменту начала тестирования, а по завершении тестирования тот же диспетчер должен составлять списки проблем с их приоритетами и передавать все материалы в команду разработки.

Часто планы тестирования требуют тестирования совместимости для нескольких конфигураций, например, Wintel PC, MAC, Linux и PDA, каждая из которых имеет множество периферийных устройств типа сканеров, принтеров, модемов, дисководов CD-RW, видеокамер, джойстиков и удаленных клавиатур или игровых устройств ввода/вывода. Кроме того, тестовое оборудование может потребоваться для тестирования нескольких проектов, поэтому между сеансами тестирования возникает необходимость в сборке или разборке конфигураций. С целью оптимизации и эффективного управления лабораторией тестирования создается должность диспетчера тестирования, которому часто помогают несколько техников этой лаборатории. Диспетчер дает указания техникам по установке тех или иных аппаратных и программных платформ и сетевых ресурсов, задействованных в нескольких проектах. Он же планирует время работы лаборатории и обслуживающего персонала. В подразделениях информационных технологий эта задача может оказаться очень сложной и должна выполняться несколькими диспетчерами тестирования. При этом должны использоваться компьютерные системы на базе мейнфреймов и множества терминалов или ПК для бизнес-аналитиков, которые соединяются между собой через глобальную корпоративную сеть. Кроме того, для имитации реальной производственно^ среды корпоративная сеть должна иметь соединения электронного обмена данными (electronic data interchange, EDI) со множеством серверов бизнес-партнеров; подобные соединения получили название телекоммуникаций типа "бизнес-бизнес" (business-to-business, B2B).

## Базы данных материалов совместного использования

Когда команда тестирования обнаруживает проблему в программном обеспечении, последняя должна быть зарегистрирована, должен быть определен ее приоритет, установлен календарный график ее устранения командой разработки, определен график подготовки версии к тестированию, выполнено повторное тестирование на предмет устранения первоначальной проблемы и возникновения побочных эффектов. Простейший способ решения этих проблем заключается в использовании базы данных материалов совместного использования. Каждая запись в этой базе данных представляет одну проблему и имеет поля, заполняемые каждым сотрудником, который сталкивался с данной проблемой. Для обеспечения эффективного поиска нужных материалов база данных материалов совместного использования может быть отсортирована и разбита на отдельные базы данных по каждой программной подсистеме, по штату разработки каждой программы, по используемому программному обеспечению других организаций и по каждому типу проблем. Кроме того, администраторы базы данных должны часто выполнять вставку обновленных записей в базу материалов совместного использования.

Для проверки того, что проблемы отслеживаются, а сведения о них сообщаются всеми организациями-участниками разработки, необходимо выполнять периодические просмотры и оценки состояния базы данных. Как правило, команды разработки сопровождают несколько версий результирующего продукта. В некоторых случаях сопровождение или создание версии может выполняться отдельным лицом, называемым создателем версии. Для этих версий должны составляться календарные планы и предусматриваться повторное тестирование, при этом база данных материалов совместного использования будет обновляться текущими данными таких сеансов повторного тестирования.

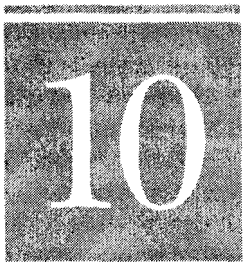
Современной реализацией базы данных материалов совместного использования должен быть защищенный Web-сайт, управляемый базой данных, который находится в экстрасети, принадлежащей генеральному подрядчику. Для обеспечения возможности обновления и внесения изменений в соответствующие записи базы данных со стороны персонала каждого проекта потребуется создать программы обслуживания приложений (application service program, ASP) с механизмом транзакций. Подобная реализация обеспечивает полностью авторизованное сотрудничество партнеров по разработке, поддерживая отсортированные по приоритетам списки проблем. Именно поэтому описанная система рассматривается как успешная реализация концепции быстрого тестирования.

## Резюме

В этой главе был исследован набор реальных технологий, которые тестирующая организация должна немедленно взять на вооружение, если она желает ступить на путь быстрого тестирования. На основании главы 7 можно сделать вывод, что сильным аргументом в пользу применения технологий быстрого тестирования служит возможность завершения разработок вдвое быстрее при использовании вдвое меньшего количества людей и при допуске вчетверо меньшего количества скрытых дефектов в

конечном продукте. Основная идея этой главы, посвященной статическому тестированию, состоит в том, что в рамках жизненного цикла разработки необходимо как можно раньше приступать к поиску ошибок и вводу отчетов об ошибках в базы данных совместного использования. Точно так же быстро следует исправлять ошибки и извлекать соответствующие уроки, помещая сведения об ошибках в контрольные перечни и модели надежности. В конце концов, эффективность проделанной работы должна оцениваться, прежде всего, на основе количества обнаруженных ошибок. Постоянное совершенствование процесса разработки приводит к снижению числа допускаемых просчетов, уменьшает количество персонала, необходимого для этапов динамического тестирования и снижает количество рекламаций со стороны клиентов.

# Технологии динамического тестирования и советы




**Темы, рассматриваемые в главе:**

- Функциональное тестирование и анализ
- Разделение по классам эквивалентности
- Анализ граничных значений
- Отрицательное тестирование
- Тестирование на основе определения степени риска
- Определение полноты охвата ветвей при тестировании
- Тестирование случаев использования
- Псевдоотладка/видоизменение
- Трассировка/трассировка снизу вверх/ мгновенные дампы/постпечать
- Создание точек прерывания/правки
- Тестирование потока данных
- Тестирование на предмет утечек памяти
- Тестирование интерфейса "человек-компьютер"
- Тестирование нагрузочной эффективности
- Тестирование конфигурации платформы
- Резюме

У читателей может возникнуть вопрос: "Какие ошибки могут остаться незамеченными после интенсивного применения технологий статического тестирования, типичных для методики быстрого тестирования?" Ответ очень прост: скрытые ошибки. Скрытые ошибки — это ошибки, которые существуют, но не обнаружены. В среднем программы, попадающие к конечным пользователям, содержат от 0,2 до 20 скрытых ошибок на тысячу инструкций исходного кода (K delivered source instructions, KSDI). В случае интенсивного применения технологий статического тестирования при разработке проекта, к началу динамического тестирования количество скрытых ошибок должно снизиться на 65—80%. Остальные 20—35% скрытых ошибок должны быть обнаружены на этапах тестирования модулей (ТМ), комплексных испытаний (КИ), системных испытаний (СИ), приемочных испытаний (ПИ) и сопровождения.

В течение динамического тестирования обнаруживаются ошибки, которые были допущены во время процессов определения требования технического задания (ТЗ), эскизного проектирования (ЭП), рабочего проектирования (РП) и кодирования. Еще? одним, иногда упускаемым из виду, источником скрытых ошибок является повторно используемое программное обеспечение. К возможными типам ошибок относятся: логические ошибки, простые опечатки, ошибки организации данных, ошибки, влияющие на эффективность, ошибки применимости, ошибки запросов баз данных, ошибки доступа к файлам, дефекты интерфейсов, ошибки управления загрузкой, отсутствующие функции, ошибочные алгоритмы, проблемы обмена данными, дефекты сценариев тестирования, неверное вычисление результатов тестов и множество других. Короче говоря, последовательное накопление ошибок, которые все еще остаются скрытыми на момент начала динамического тестирования, происходит на всех предыдущих этапах проектирования. За счет применения методики быстрого тестирования, которая позволяет максимально быстро выявить и исправить большинство скрытых ошибок, можно сэкономить время и существенно снизить трудозатраты.

На этапах ТЗ и ЭП жизненного цикла разработки функциональные требования объединяются с нефункциональными. Примерами нефункциональных требований могут служить календарный план поставки продукта, состав установочного комплекта, требования по заполнению базы данных, временные характеристики алгоритма, человеческий фактор, требования к поддерживаемым конфигурациям, коммуникационная инфраструктура, требования к обеспечению безопасности, надежность и т.п. Большинство из этих требований разрабатывается на основе стандартов программирования или руководств по стилю, действующих в конкретной организации. Как правило, стандарты программирования создаются на базе опыта работы организации. Извлеченные в процессе деятельности уроки, наряду с результатами маркетинговых исследований программной продукции конкурирующих организаций, оформляются в виде стандартов программирования, иногда называемых руководствами по стилю. Если ваша организация готова инвестировать определенные средства в тестирование, имеет смысл вкладывать их в автоматизацию тестирования стандартов программирования или руководств по стилю. После такого инвестирования персонал, занимающийся быстрым тестированием, может повысить скорость работы за счет многократного использования существующих планов тестирования, тестовых случаев, сценариев и средств тестирования, которые разработаны для проверки выполнения нефункциональных требований. Однако специалистам по тестированию все еще придется разрабатывать тестовые случаи, сценарии тестирования,

методы обработки результатов тестов и средства тестирования при подготовке к испытаниям новых функций продукта, т.е. выполнения функциональных требований.

В этой главе исследуется множество технологий динамического тестирования. Тем не менее, представленный в ней набор технологий не является исчерпывающим. Цель применения этих технологий заключается в планомерном и эффективном уменьшении количества скрытых ошибок в программном продукте. Во время планирования своей работы члены команды тестирования должны самостоятельно определять подходящий количественный и качественный состав применяемых технологий. Результатами выполнения задач по разработке тестов, которые имеют исключительно большое значение для достижения цели планомерного выявления скрытых ошибок, являются планы тестирования, тестовые случаи, сценарии тестирования и методы обработки результатов тестирования.

## Функциональное тестирование и анализ

Функции, на которые иногда ссылаются как на функциональные возможности, — это именно то, разработку чего оплачивают пользователи. Часто в литературе, посвященной торговле и маркетингу, эти функции представляют в виде маркированных перечней, которые сравниваются с аналогичными перечнями конкурирующих продуктов. Эти перечни функций продукта определяются на этапе разработки требований и тщательно отслеживаются в ходе выполнения этапов проектирования и кодирования, например, в матрице прослеживаемости требований (Requirements Traceability Matrix, RTM). Эти функции являются побочным результатом так называемого пошагового уточнения, и они представляют собой фрагменты исходного кода, образующие модули или объекты.

Функциональный анализ представляет собой действия по описанию характеристик функции, влияющих на процесс ее проектирования и тестирования. Предположим, что имеется разложение в ряд Тейлора для функции y(x;)=sin(x) (см. раздел "Символьное выполнение" в главе 9). На этапах проектирования следовало бы выполнить функциональный анализ для определения точности вычисления разложения как в плане указания размера слова, требуемого для хранения используемых переменных с плавающей точкой, так и в плане любых промежуточных результатов. Например, для некоторых функций можно показать, что точность частичного разложения для переменной *y(x)* будет повышаться с увеличением количества членов ряда Тейлора. Важно отметить, что если на этапах проектирования функциональный анализ не выполнялся, его придется выполнить во время подготовки к динамическому тестированию. Организации, откладывающие оценку точности алгоритма до момента начала динамического тестирования, не могут быть отнесены к тем, которые исповедуют стратегию быстрого тестирования. В организациях, взявших на вооружение методику быстрого тестирования, во время разработки тестовых случаев, подготовки тестовых данных, сценариев тестирования и технологий обработки результатов тестов специалисты, ответственные за подготовку к динамическому тестированию, просто ссылаются на документацию по функциональному анализу.

**ПРИМЕР: НЕОБНАРУЖЕННОЕ ПЕРЕПОЛНЕНИЕ ПРИ ВЫПОЛНЕНИИ ОПЕРАЦИЙ С ПЛАВАЮЩЕЙ ТОЧКОЙ (ГЭРИ КОББ)**

**Автор вспоминает ситуацию, когда ему довелось столкнуться с реализацией метода конечных разностей для решения системы дифференциальных уравнений в частных произj водных, причем пришлось вплотную заняться функциональным анализом влияния округления и усечения на промежуточные результаты. В ходе этого исследования выяснилось что исходный код вычисляет интеграл по сетке значений переменной с именем mass (масса). По мере добавления отдельных масс, промежуточный результат становился на столько большим, что при достижении приблизительно середины сетки добавление, остальных масс переставало влиять на значение суммы, которое.представлялось значени ем с плавающей точкой. Это было связано с быстрым увеличением показателя степени промежуточной суммы, что приводило к тому, что инструкция выравнивания |показателя степени (используемая при работе со значениями с плавающей точкой) компьютера пренебрегала массами, показатели степени которых были малы по сравнению с показателем степени промежуточной суммы. К упомянутому функциональному анализупри ШЛось прибегнуть, когда тестирование позволило сделать вывод что программа будет| выдавать различные результаты при вылолнении ее с использованием инструкции работы со значениями с плавающей точкой одинарной и двойной точности либо при Запуске на компьютерах, в среде которых длины слов отличаются.**

# Разделение по классам эквивалентности

Принадлежность двух элементов данных к одному и тому же классу эквивалентности просто означает, что с каждым из них функция выполняет одни и те же операции. Принадлежность двух элементов данных к различным классам эквивалентности означает, что существует, по меньшей мере, одна строка кода, требуемая для обработки одного элемента данных, которая не будет использоваться при обработке другого элемента данных. Часто данные, принадлежащие к одному из двух классов эквивалентности, называют *правильными данными,* а данные второго класса эквивалентности — *неправильными данными* для данной функции. Ветви кода, которые используются для обработки правильных данных, называются *удачными ветвями,* в то время как ветви, выполняемые функцией при обработке неправильных данных, называются *неудачными ветвями.* В проектной документации для большинства функций определены *правильные* и *неправильные* входные данные. Для определения классов эквивалентности данных для каждой функции тестировщики должны прочесть проектную документацию. Если эта информация отсутствует в проектной документации, тестировщику придется применить функциональный анализ и восстановить информацию о классах эквивалентности снизу-вверх. В некоторых случаях в проектной документации может использоваться также термин *допустимых* и *недопустимых* данных для дайной функции. Операции тестирования во время ввода неправильных данных достаточно точно называются отрицательным тестированием. Неправильные данные выбираются с тем, чтобы убедиться в наличии в каждой функции обработчиков исключений, выполняющих обработку неправильных данных. Отрицательное тестирование не ограничивается одним лишь выполнением операций тестирования для случая неправильных данных (обратитесь к разделу, посвященному отрицательному тестированию, далее в этой главе).

Одна из принципиальных отличительных черт специалиста по тестированию, применяющего технологии быстрого тестирования, состоит в том, что при оценке вероятности наличия скрытых ошибок, которые могут препятствовать эффективно-

му использованию конечной программы, он всегда учитывает широту и степень покрытия тестирования. Чтобы гарантировать тестирование как удачных, так и неудачных ветвей внутри каждой функции, тестировщик должен уделять пристальное внимание классам эквивалентности входных данных функций.

## Анализ граничных значений

Весьма перспективной областью для поиска ошибок являются границы классов эквивалентности функции. Как правило, анализ, который ведет к определению классов эквивалентности, определяет и эти границы. Включение нескольких пограничных значений в тестовые случаи для данной функции поможет проверить удовлетворение ожиданий (требований) пользователя. Кратко рассмотрим причины ошибок разработчиков при кодировании граничных значений. Причины появления в программных продуктах ошибок, связанных с граничными значениями, достаточно легко понять, анализируя количество преобразований, которые выполняются с момента разработки требований до момента начала динамического тестирования. Прежде всего, все требования создаются на высоком уровне. Они содержат не слишком много подробностей. Как правило, на этапе рабочего проектирования (РП) проектировщики программного обеспечения добавляют в RTM так называемые производные требования, преобразуя исходные требования в подробные и точные с точки зрения вычислений.

В организациях, занимающихся быстрым тестированием, уже осознали важность этапа РП для команды тестирования. Если производные требования отсутствуют или разработаны неправильно, то персоналу, занятому кодированием или тестированием, в ходе планирования, соответственно, придется дополнять высокоуровневый проект архитектуры и исходные требования этими документально оформленными подробностями рабочего проекта, в том числе и определяющими поведение программы на границах классов эквивалентности. В качестве примера рассмотрим конструкции IF. По прошествии ряда лет многие исследователи в области компьютерных наук отметили, что утверждения в конструкции IF — это те программные элементы, в которых ошибки встречаются наиболее часто. Решение об использовании условия "меньше чем" или "меньше или равно" в утвердительной части конструкции IF часто принимается на этапе РП. В противном случае отсутствие этих спецификаций должно быть выявлено во время инспекций, выполняемых на этапе РП.

## Отрицательное тестирование

Отрицательное тестирование означает всего лишь подход, при котором тестировщик, рассматривая программный продукт в качестве "черного ящика", определяет способы, как заставить продукт выдавать неверные ответы или вообще прервать выполнение. Выяснив у проектировщиков, какие требования должны быть выполнены перед запуском программы, хороший специалист по тестированию может определить набор тестовых случаев для выяснения реакции программного продукта на несоблюдение одного из этих предварительных условий. Рассматривая программу под этим непривычным углом зрения, тестировщик предпринимает попытки ее выполнения:

- на платформах, на которых ее выполнение не планировалось;
- при отсутствии коммуникационных линий или при вводе неправильных входных данных;
- при отсутствии файлов данных, при отсутствии записей в базах данных или при произвольно переставленных данных в файлах данных;
- при неверно введенных именах ссылок или при неопределенных, неправильных или отсутствующих конфигурационных параметрах;
- при выключенных периферийных устройствах типа принтеров, сканеров, внешних дисководов компакт-дисков или CD-RW, внешних жестких дисков, внешних динамиков и т.п.

Как уже упоминалось в разделе "Разделение по классам эквивалентности", к отрицательному тестированию относится также и ввод *неправильных* данных. Эти *неправильные* данные могут поступать в форме недопустимых данных, введенных пользователем, случайно заполненных данными коммуникационных буферов, недопустимых значений в индексных файлах, переполненных журнальных файлов, в которых указатель находится в конце файла, и т.д.

Устойчивость программы — это ее способность выдержать без сбоя отрицательное тестирование. Естественно, существует определенный предел устойчивости, тем не менее, разработчики программного обеспечения должны критично относится к коду и тестировать его как с правильными, так и с *неправильными* данными. Они должны предусмотреть самопроверку на предмет присутствия минимально допустимой системной конфигурации и ее готовности выполнить приложение. Это самотестирование должно предприниматься в начале выполнения программы, поскольку с момента установки продукта конфигурация могла измениться. Недостаточно выполнять самотестирование конфигурации только во время установки приложения.

Персонал, занимающийся тестированием, располагает широким спектром конфигураций оборудования, на котором может выполнять тестирование системы. С персоналом разработки, который использует типовые компьютеры, дела обстоят иначе. Важно, чтобы все члены команды тестирования осознали свою ответственность за проверку того, что продукт работает и дает одинаковые результаты во всех конфигурациях, которые указаны в документе требований. Помимо обязательных конфигураций, персонал, ответственный за тестирование, должен выполнить проверку обязательных конфигураций, поддержка которых была отключена или которые были как-то расширены. Если во время тестирования эти конфигурации приводят к генерации исключений, то, как минимум, документация по продукту должна содержать предупреждение о недопустимых конфигурациях.

Стратегия отрицательного тестирования никогда не должна напоминать стрельбу наудачу. Напротив, планы тестирования должны быть тщательно продуманными, непротиворечивыми, завершенными и эффективно обеспечивать обнаружение ошибок. Штат, занятый разработкой методов тестирования, должен быть полностью укомплектован для разработки тестовых случаев, обеспечивающих соблюдение стратегии тестирования, которая задокументирована в плане тестирования. При соблюдении этих условий тестирование системы будет эффективно в плане обнаружения и устранения в продукте множества ошибок.

# Тестирование на основе определения степени риска

Еще одна технология быстрого тестирования заключается в определении меры степени риска и введении соответствующих политик и процедур для ее использования. В ходе всей разработки проекта мера степени риска должна применяться абсолютно единообразно в соответствии с положениями, задокументированными в политиках и процедурах. Мера степени риска может определяться в какой-либо строгой форме, как показано в таблице 10.1.

**Таблица 10.1. Мера степени риска**

| Уровень риска | Стандарт для выбора этого уровня |
|---|---|
| 1 | Ошибка приводит к прерыванию всех приложений и сеансов коммуникации; требуется перезапуск, поэтому данные сеанса могут быть утеряны или оказаться неполными. |
| 2 | Ошибка приводит к неправильным ответам, которые могут влиять на последующие результаты, если пользователь продолжает работать с приложением. |
| 3 | Ошибка позволяет получить правильные результаты, но требования пользователя удовлетворяются не полностью. |
| 4 | Информация, представляемая пользователю, или эффективность приложения не отвечает требованиями по применимости. |
| 5 | Данное свойство может быть улучшено, но оно не оказывает отрицательное влияние на работу пользователя. |

Для каждой подсистемы конечного программного продукта организуется группа контроля за внесением изменений, занимающаяся сопровождением, отслеживанием состояния и утверждением версий, в которых устранены отдельные поднаборы зафиксированных ошибок. В организациях часто ведется база данных ошибок, которая существует под эгидой группы контроля за внесением изменений, но используется совместно (с предоставлением доступа только по чтению) командами разработки и тестирования. Персонал этих команд исправляет и повторно тестирует ошибки, запланированные для устранения в будущих версиях конечного продукта. Эта инфраструктура поддерживает подход к быстрому тестированию на основе определения степени риска, который может обеспечить поэтапную передачу конечного продукта заказчику. Инфраструктура подходит также для двух партнеров по разработке программного обеспечения, географически расположенных в различных точках земного шара.

Стандарт IEEE Standard 1044 института инженеров по электротехнике и электронике (Institute of Electrical and Electronics Engineers — IEEE) [24] устанавливает дополнительные определения, действия и процессы во время отслеживания ошибок. Этот стандарт возлагает на тестировщиков ответственность за классификацию ошибки во время ее обнаружения (распознавание). Ошибка, о которой сообщается в документации по тестированию, анализируется на предмет оказываемого ею влияния, и для нее предлагается значение меры степени риска (влияние распознавания).

Команда разработки или сопровождения предпринимает шаги (исследование), чтобы удостовериться в повторяемости ошибки, и пытается определить ее основную причину. Затем делаются уточнения предложенного значения меры степени риска **и** плана, в том числе выделенных ресурсов, первой версии, в которой данная ошибка будет устранена, и оцениваются затраты (влияние исследования). В ходе выполнения этого плана предпринимаются действия по корректировке, и команда тестирования снова обращается к тестовому случаю, во время прогона которого ошибка была обнаружена впервые. Кроме того, команда выполняет набор регрессивных тестов, чтобы удостовериться в том, что исправление ошибки не привело к появлению других ошибок. На основе результатов повторного тестирования заключительные выводы документируются в примечании по реализации той версии, которая вначале содержала ошибку. Краткое описание этого процесса приведено **в** таблице 10.2

Диаграмма информационных потоков, приведенная на рис. 10.1, помогает упорядочить подпроцессы на каждом из этапов и увязать их со всем процессом. Типовые группы контроля за конфигурацией **и** технологическим циклом, которые в настоящее время используются во многих организациях, могут обеспечить удобное сопровождение этого процесса отслеживания ошибок. Для того чтобы закрыть проблему, связанную с ошибкой, потребуется согласовать **и** программу, и ее документацию, в том числе требования, эскизную и рабочую документацию по проекту, а также руководство пользователя.

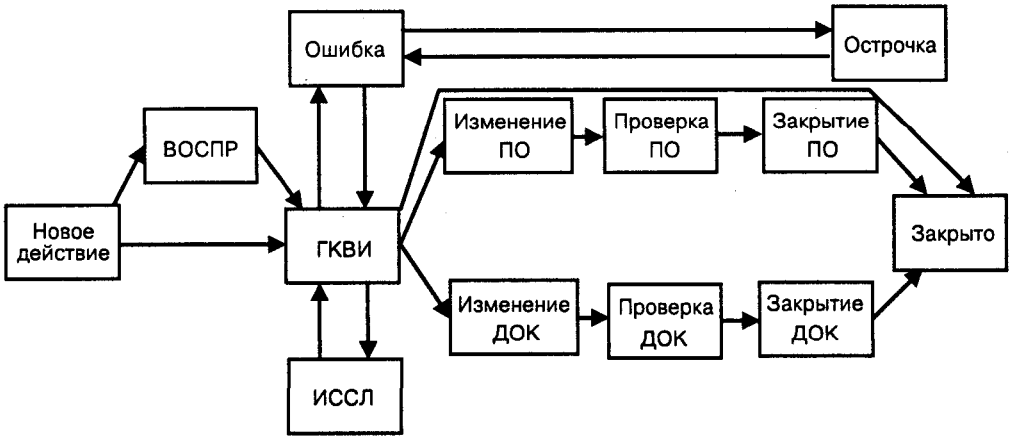



***Рис. 10.1. Информационные потоки устранения ошибки в соответствии со стандартом IEEE Standard 1044.***

**Таблица 10.2. Терминология процесса отслеживания ошибок в соответствии со стандартом IEEE Standard 1044.**

| | | |
|---|---|---|
| Распознавание | Выполняемые действия | Какие действия выполнялись во время проявления ошибки?<br>На каком этапе жизненного цикла продукта она была обнаружена? |
| | Предполагаемая причина | Какова может быть причина ее появления? |
| | Повторяемость | Удалось ли повторно добиться появления ошибки? |
| | Симптомы | Каким образом проявляется ошибка? |
| | Состояние продукта | Какова возможность использования продукта? |
| Влияние распознавания | Значение для клиента | Насколько устранение ошибки важно для клиента? |
| | Степень серьезности | Насколько серьезной была ошибка? |
| | Влияние на выполнение функций/ безопасность | Насколько серьезной была ошибка с точки зрения выполнения стоящих перед проектом целей или угрозы для безопасности людей? |
| Исследование | Действительная причина | Что привело к возникновению ошибки? |
| | Источник | Что было источником ошибки? |
| | Тип | Каким был тип ошибки? |
| Влияние исследования | Влияние на календарный план | Сравнительное влияние на календарный план разработки продукта. |
| | Влияние на стоимость | Сравнительное влияние на бюджет. |
| | Степень риска проекта | Риск, связанный с устранением ошибки. |
| | Влияние на качество | Влияние на качество продукта или надежность устранения ошибки. |
| | Социальные последствия | Социальные последствия устранения ошибки. |
| | Приоритет | Степень важности устранения ошибки. |
| Действие | Решение проблемы | Какие действия были предприняты перед закрытием отчета? |
| | Корректирующее действие | Какие действия следует предпринять для предотвращения появления этой ошибки в будущем? |
| Заключительные выводы | Заключительные выводы | Какова судьба ошибки, исходя из заключительного анализа? |

# Определение полноты охвата ветвей при тестировании

В области проектировании программного обеспечения существует несколько определений тестирования ветвей. Наиболее традиционное определение гласит, что ветвь — это последовательность выполнения операторов исходного кода, начинающаяся с точки входа или условного разветвления и заканчивающаяся следующим условным разветвлением или точкой выхода. Такие ветви были названы DD-ветвями (от decision-to-decision path — ветвь типа "решение-решение"). Теоретики подхода быстрого тестирования признают бесперспективность попыток обеспечить при тестировании полный охват всех DD-ветвей и рекомендуют вместо этого использовать программное средство автоматизации документирования всех DD-ветвей в исходном коде. Кроме того, на этапе тестирования блоков или комплексных испытаний необходимо определить приоритеты тестирования, чтобы испытания обязательно охватывали поднабор DD-ветвей, в которых вероятность наличия ошибки наиболее высока.

Листинг программы, приведенный на рис. 10.2, был создан в период 1979-1981г.г., когда авторы разрабатывали систему тестирования программного обеспечения (Software Testing System, STS). Эта система предназначалась для внутреннего использования в корпорации Texas Instruments, Inc., основанной организацией Advanced Software Technology (AST). STS была программным инструментом, который позволял программистам на языке Fortran выполнять перечисление ветвей исходного кода. Для тестировщика STS служила удобным средством определения нужных значений переменных программы для перемещения программного счетчика по конкретной DD-ветви. STS могла устанавливать зонды в каждой DD-ветви. Во время выполнения эти зонды регистрировали номера DD-ветвей по мере их выполнения. После нескольких запусков программы анализ журнальных файлов позволял выяснить, какие DD-ветви не выполнялись. В случае аварийного прерывания выполнения программы журнальный файл, содержащий трассировку ветвей, объединялся с файлом с пронумерованными строками исходного кода, что позволяло генерировать отчеты, содержащие строки исходного кода в порядке, обратном выполнению. Этот файл был очень полезен во время отладки, поскольку его можно было загрузить в текстовый редактор и выполнять поиск имен переменных, которые были связаны с прерыванием программы.

```
ADVANCED SOFTWARE TECHNOLOGY — FORTRAN SOFTWARE TESTING SYSTEM — (STS)

    DATE: 12/08/80                TIME: 12:40:34                PAGE:    1

THE OPTIONS IN EFFECT FOR THIS RUN OF STS STATIC ANALYZER ARE:

        LIST       = INPUT
        SCANONLY   = NO
        APPEND     = NO
        TYPE       = BOTH
        TIMEFIO    = NO
        DOPTION    = NO
        COPY       = YES
```

```
1* C     THIS PROGRAM WILL SPLIT A FILE WITH <REP CARDS IN IT TO A DIRECTORY
2* C     AND LIST THE MEMBER NAMES OF ALL THE FILES CREATED.
3* C
4* C     UNIT5 — INPUT  (PARM1)
5* C     UNIT6 — OUTPUT DIRECTORY  (PARM2)
6* C     UNIT7 — LIST  (PARM3)
7* C
8*              IMPLICIT INTEGER*2 (A-Z)
9*              LOGICAL HEQ,HNE,FIRST,NEWMEM,GETYNO,ASCII
10*             DIMENSION
                CARD(80),PATH80(80),PATH40(40),CHARS(4),OLDMEM(4)
11*             DATA CBB,FIRST,ASCII/2H ,.TRUE.,.FALSE./
12*             DATA LESSTH,R,E,P/2H< ,2HR ,2HE ,2HP /
13* 1000        FORMAT(80A1)
14*             CALL SCIINT(IERR)
15* C
16* C           OPENFH LUNO,PARM,ACCESS,BKSZ )
17* C
18*             CALL OPENFL(5,1,1,80)
19*             CALL OPENFL(7,3,3,80)
20*             ASCII=GETYNO(4,IERR)
21*             CALL BANNER(ASCII)
22*             CALL GETDIR(PATH80,NDEX,2)
23*             NDEXM2=NDEX-2
24*            WRITE(7,5000)  (PATH80(I),1=1,NDEXM2)
25*             WRITE(7,6000)
26* 5000       FORMAT('  THE OUTPUT DIRECTORY',/,IX,79A1)
27* 6000       FORMAT!//,'  FILE WITHIN OUTPUT DIRECTORY NUMBER
                RECORDS')
28*             NRECS=0
29*             RECCNT=0
30*
31*
32*
33*
34* 1           CONTINUE
35*             DO INDEX 1=1,80
36*             CARD(I)=CBB
37*             END DO
38*             READ(5,1000,END=100) CARD
39*             IF(ASCII) CALL EBCDIC(2,CARD,0)
40*             C WRITE(10,1000) CARD
41*              NEWMEM=(CARD(1).EQ.LESSTH.AND.CARD(2).EQ.R.AND.CARD(3).EQ.E.

                MODULE NAME — MAIN
ADVANCED SOFTWARE TECHNOLOGY — FORTRAN SOFTWARE TESTING SYSTEM —(STS)
    DATE: 12/08/80                  TIME: 12:40:44                PAGE:

42*             &AND.CARDH) .EQ.P)
43* C           WRITE(10,111) CHARS,NDX,LEN
44* 111         FORMATC AT 111 '4A2,2X,2110)
45*              IF(FIRST.AND.(.NOT.NEWMEM))
46*              THEN
47*              WRITE(7,1500)
```

```
48* 1500       FORMAT(' ERROR: FIRST RECORD IS NOT A <REP CARD')
49*            CALL ENDFIL(7)
50*            STOP
51*            END IF
52*            IF(NEWMEM)
53*            THEN
54*            NDX=5
55*            CALL  HFIELD(CARD,NDX,CHARS,80,LEN,CODE)
56*            IF (.NOT.FIRST)
57*            THEN
58*            CALL ENDFIL(6)
59*            NRECS=NRECS+RECCNT
60*            WRITE(7,7000)OLDMEM,RECCNT
61* 7000       FORMAT(10X,4A2,26X,15)
62*            RECCNT=0
63*            END IF
64*            DO INDEX 1=1,4
65*            OLDMEM(I)=CHARS(I)
66*            END DO
67*            CALL  SETMEM(CHARS,PATH80,NDEX,PATH40,3,6)
68*            FIRST=.FALSE.
69*            DO INDEX 1=1,80
70*            CARD(I)=CBB
71*            END DO
72*            READ(5,1000,END=100)  CARD
73*            IF(ASCII)  CALL EBCDIC(2,CARD,0)
74* C          WRITE(10,1000)  CARD
75*            END IF
76*            WRITE(6,1000)(CARD(I),1=1,80)
77*            RECCNT=RECCNT+1
78*            GO TO 1
79* 100        CALL CLOSEW(5,IERR)
80*            CALL ENDFIL(6)
81*            NRECS=NRECS+RECCNT
82*            WRITE(7,7000)OLDMEM,RECCNT
83*            WRITE(7,3000)  NRECS
84* 3000       FORMAT(//,'THE TOTAL NUMBER OF RECORDS SPLIT WAS:',15)
85*            CALL ENDFIL(7)
86*            STOP
87*            END
(STS)  -- STATIC ANALYSIS FOR MODULE "MAIN ", BEGINNING AT LINE 14

         PATH   1:   14        THRU37         EOP
         PATH   2:   37        JUMP35         THRU37      EOP

         MODULE NAME -   MAIN
ADVANCED SOFTWARE TECHNOLOGY - FORTRAN SOFTWARE TESTING SYSTEM -  (STS)
    DATE: 12/08/80                    TIME: 12:40:56                  PAGE:

         PATH   3|  37        THRU38         EOP
         PATH   4|  38        JUMP79         THRU86      EXIT
         PATH   5|  38        THRU39         EOP
         PATH   6|  39    A   THRU39      B  THRU45      EOP
         PATH   7|  39    A   THRU45         EOP
```

```
         PATH   8    45        JUMP52          EOP
         PATH   9    45        THRU50          EXIT
         PATH  10    52        JUMP76          THRU78      JUMP34      THRU37
                                 EOP
         PATH  11    52        THRU56          EOP
         PATH  12    56        JUMP64          THRU66      EOP
         PATH  13    56        THRU66          EOP
         PATH  14    66        JUMP64          THRU66      EOP
         PATH  15    66        THRU71          EOP
         PATH  16    71        JUMP69          THRU71      EOP
         PATH  17    71        THRU72          EOP
         PATH  18    72        JUMP79          THRU86      EXIT
         PATH  19    72        THRU73          EOP
         PATH  20    73     A  THRU73       B  THRU78      JUMP34      THRU37
                                 EOP
         PATH  21    73     A  THRU78          JUMP34      THRU37      EOP

STS CYCLOMATIC COMPLEXITY INTERVAL    ( 11 ,

         MODULE NAME - MAIN
ADVANCED SOFTWARE TECHNOLOGY  -- FORTRAN SOFTWARE TESTING SYSTEM  •- (STS)
    DATE: 12/08/80                    TIME: 12:40:59 .                PAGE:

88*             SUBROUTINE BANNER(ASCII)
89*             IMPLICIT INTEGER*2  (A-Z)
90*             LOGICAL ASCII
91*             DIMENSION DATI(8),NDX(1),VAL(2,2)
92*             DATA NOPTS,VAL/1,2H Y,2H N,2HES,2HO  /
93*             DO  INDEX  I=1,NOPTS
94*             NDX(I)=2
95*             END DO
96*             IF(ASCII)NDX(1)=1
97*             CALL DT(DATI)
98*             WRITE(7,1000)DATI
99*             WRITE(7,2000)  ((VAL(NDX(I),J),J=1,2),1=1,NOPTS)
100*            RETURN
101* 1000       FORMAT('  SPLIT UTILITY',/,
102*            &•DATE:',4A2,',TIME:'4A2,
103*            & //,' OPTIONS SELECTED')
104* 2000       FORMAT(' CONVERT EBCDIC->ASCII=',2A2,/)
105*            END
 (STS)  — STATIC ANALYSIS FOR MODULE "BANNER ",  BEGINNING AT LINE  93

         PATH  22    93        THRU95          EOP
         PATH  23    95        JUMP93          THRU95      EOP
         PATH  24    95        THRU96          EOP
         PATH  25    96     A  THRU96       B  THRU100     EXIT
         PATH  26    96     A  THRU100         EXIT

         STS CYCLOMATIC COMPLEXITY INTERVAL =  (   3  ,   3  )
THIS IS A NORMAL COMPLETION OF STS  --  STATIC ANALYZER RELEASE 3.1
```

*Рис. 10.2. Автоматизированный вывод листинга ветвей и значений показателей цикломатической сложности, выполняемый системой STS.*

Для того чтобы можно было прочесть листинг ветвей, потребуется ввести следующие определения:

- <номер строки> — номер, помещаемый программой в каждую строку кода
- JUMP <номер строки> означает какую-либо форму конструкции GO TO, ENDDO или IF
- THRU <номер строки> включает все строки вплоть до строки <номер строки>
- <номер строки> A означает часть THEN конструкции IF
- <номер строки> B означает часть ELSE конструкции IF
- EOP указывает конец ветви

Кроме того, при чтении этого листинга необходимо обратить внимание на пару вычисленных и отображенных значений показателя цикломатической сложности. Число слева определено Томасом Дж. Маккейбом (Thomas J. McCabe) в [31]. Значение справа — это показатель цикломатической сложности, определенный Гленфордом Дж. Майерсом (Glenford J. Myers) [37] в ответ на первоначальную публикацию Маккейба. Вспомните, что показатель цикломатической сложности — это минимальное количество отдельных запусков модуля, которое потребуется для хотя бы однократного выполнения каждой строки кода. Данная информация может оказаться полезной при прогнозировании затрат на тестирование. Можно попытаться найти все ветви в одном из таких модулей и прочувствовать ценность описываемого программного средства с точки зрения обеспечиваемой им экономии времени.

Большинство теоретиков программирования сходятся во мнении, что свыше половины ошибок в новом коде являются результатом ошибок в логике построения кода. Тестирование ветвей выявляет логическую структуру кода и позволяет тестировщику сосредоточить внимание на ее правильности. Однако арсенал специалиста по тестированию не должен ограничиваться только одним этим средством, поскольку строки кода в части THEN конструкции IF всё же могут содержать ошибки, которые приводят к получению неправильных результатов или аварийному прерыванию программы.

Кроме того, следует учитывать, что многие ветви в программе являются *тупиковыми*. Это понимают очень немногие тестировщики. Как правило, разработчики, создающие отладочный код, управляют его выполнением при помощи переменной, значение которой по умолчанию устанавливается таким, что отключает отладку. С момента поставки кода и в течение всего времени его использования клиентом это значение по умолчанию остается в состоянии отключенной отладки. В следующем разделе приведена схема снижения трудоемкости выполнения полного тестирования DD-ветвей, основанная на использовании приоритетов.

# Тестирование случаев использования

В наше время в объектно-ориентированном программировании часто используется технология проектирования, называемая проектированием случаев использования (см. рис. 10.3). В случаях использования отражается взаимодействие отдельных элементов друг с другом на основе конкретного сценария. Популярные сценарии случаев использования находят свое отражение в большинстве проектов. Строгое отображение сценариев на случаи использования позволяет сделать вывод о существовании

набора популярных случаев использования в рамках различных проектов. В [35] была отмечена такая взаимосвязь и предпринята попытка связать данные частоты использования с диаграммами случаев использования. Данные частоты использования могут быть выражены в процентах от общего количества случаев использования. Имея такую информацию о проекте продукта, специалисты по быстрому тестированию могут выделять более 50% времени, в течение которого создаются тестовые случаи, на менее чем 50% кода, характеризующегося самым частым применением, т.е. на популярные случаи использования.

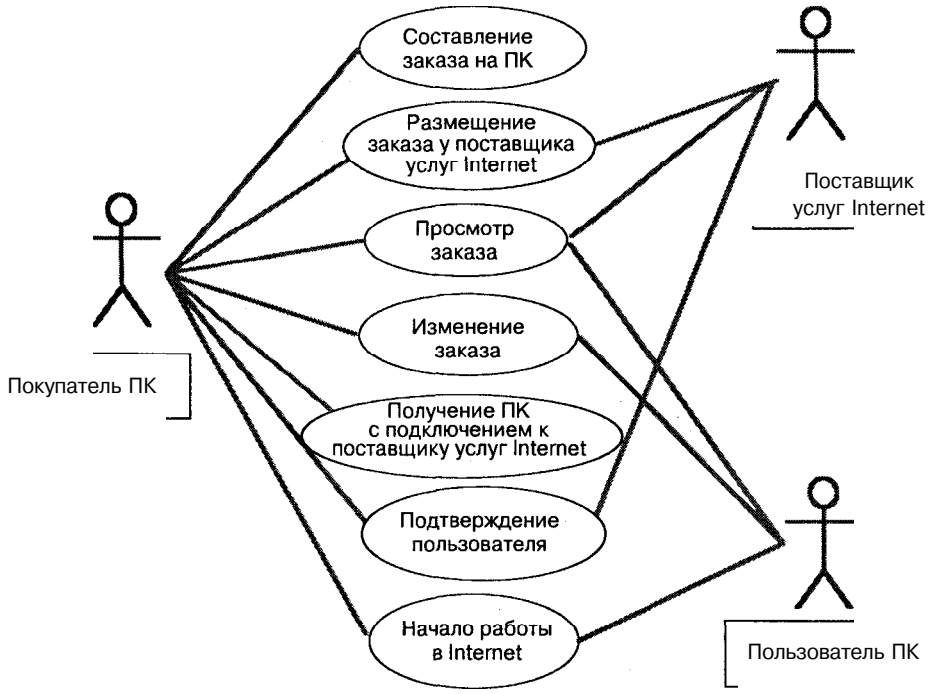



*Рис. 10.3. Пример диаграммы случаев использования - регистрация у поставщика услуг Internet во время приобретения ПК.*

Другой аналогичный подход заключается в зондировании DD-ветвей, как описывалось в предыдущем разделе. После того как зонды вставлены в код, необходимо выполнить большой объем тестовых случаев и отсортировать полученные журнальные файлы выполнения ветвей по номерам зондов. Количество тестов должно определяться исходя из требований, информационных потоков и возможного опыта использования в технологическом цикле существующих систем. DD-ветви, номера зондов которых встречаются наиболее часто, образуют эмпирический прогноз частот основного использования. Описанный процесс относится к технологии быстрого тестирования, поскольку его назначение состоит в экономии времени и повышении эффективности обнаружения ошибок. Что может быть хуже, чем когда несколько специалистов по тестированию тратят массу времени на разработку тестовых случаев для нескольких ветвей нового фрагмента кода, лишь только для того, чтобы выяс-

нить, что код является *невыполняемым* и поэтому не может быть протестирован. Или, скажем, в принципе новый код является *выполняемым,* но выполняется только при удовлетворении какого-то маловразумительного условия. Какое бессмысленное расходование времени и ресурсов с нулевой эффективностью, поскольку ни одна ошибка не обнаруживается!

Проектирование случаев использования и прогнозирование частоты использования каждого из них позволяет определить приоритеты тестовых случаев. Чтобы можно было оптимизировать ресурсы и эффективность тестирования, эти приоритеты должны быть положены в основу разработки тестовых случаев и порядка их прогона.

Прежде чем применить к тестированию подход, основанный на случаях использования, тестировщики и разработчики должны обеспечить гарантированное соответствие случаев использования списку требований и коду. Несоответствие документации по требованиям и кода — это недостаток, который часто отмечается во время аудита многих проектов по созданию программного обеспечения. В некоторых организациях документации по программе уделяется не такое пристальное внимание, как документации для пользователя. Иногда случаи использования, разработанные на основе исходных требований, передаются группе тестирования без проверки на соответствие текущим требованиям. Однако за это время исходные требования могли претерпеть изменения. В таких ситуациях тестировщики, используя информацию о случаях использования, могут получить ложное обнаружение ошибок. В ходе дальнейших исследований выяснится, что код удовлетворяет списку требований, но не случаям использования. Вероятность наличия ошибок в тестовом случае и результатах будет высокой, а усилия, потраченные на разработку тестового случая, окажутся напрасными.

## Псевдоотладка/видоизменение

*Псевдоотладка (bebugging),* являющаяся одной из форм видоизменения программы, — это способ определения эффективности стратегий тестирования, применяемых в проекте. Прежде чем приступать к псевдоотладке, необходимо заручиться поддержкой и тестировщиков, и разработчиков. Как бы вы ответили на вопрос, часто задаваемый руководителем разработки: "Когда можно ожидать выявления большинства ошибок в этой версии?". Обычно руководитель тестирования задает встречный вопрос: "А сколько мелких ошибок нам нужно найти?" Основным показателем прогресса тестирования должно быть процентное отношение числа найденных ошибок к числу скрытых ошибок, которые нужно найти. Проблема, связанная с вычислением упомянутого показателя, состоит в том, что и числитель, и знаменатель этого отношения неизвестны.

Псевдоотладка заключается в преднамеренном внесении ошибок в программу. Далее полученная версия тестируется, и после этого определяется эффективность тестирования путем сравнения обнаруженных и искусственно созданных ошибок. Отношение числа обнаруженных искусственно созданных ошибок к числу созданных ошибок должно быть равно отношению числа найденных ошибок к числу скрытых ошибок в продукте. Этот метод позволяет оценить показатель прогресса тестирования, определенный чуть выше.

Организации, в которых применяется быстрое тестирование, находят, что псевдоотладка может ускорить проведение тестирования благодаря определению условий его прекращения. Фактически, процесс преднамеренного внесения ошибок требует творческого подхода. Если искусственно созданные ошибки распределяются по типам в соответствии с исторически сложившейся моделью ошибок аналогичных разработок, с использованием того же языка программирования, и если обнаруженные ошибки таким же образом распределены по типам, то анализ прогресса тестирования можно выполнить по типам. В этом случае можно не только со всей определенностью ответить на вопрос: "Когда можно ожидать выявления большинства ошибок в этой версии?", но и сказать: "Мы нашли большинство логических ошибок в программе, а теперь заняты разработкой дополнительных тестовых случаев для проверки надежности модулей обработки ошибок". Аналогично, все искусственно созданные ошибки могут быть распределены по уровням серьезности. И тогда, если обнаруженные ошибки также распределить по уровням серьезности, можно не только ответить на заданный вопрос, но и сказать: "Мы нашли большинство ошибок с уровнями серьезности 1 и 3, а сейчас разрабатываем дополнительные тестовые случаи для обнаружения ошибок с уровнями серьезности 2 и 4".

Аналогичные подходы к видоизменению программы могут использоваться в отношении ошибок, которые не связаны с исходным кодом, в том числе неправильных данных в базе данных, некачественных коммуникационных линий, поврежденных данных, неправильных данных, вводимых пользователем, ненадежных интерфейсов между процессами и т.п. Видоизменение программы представляет собой эффективное средство, которое может использоваться разрабатывающей организацией, которая повторно использует код из другого проекта. Частью процесса повторного использования исходного кода является его сопровождение. Псевдоотладка может применяться для проверки хода этого сопровождения, определяя конечные цели тестирования, которые будут использоваться при разработке новых тестовых случаев.

## Трассировка/трассировка снизу вверх/ мгновенные дампы/постпечать

Специалисты по тестированию могут воспользоваться рядом инструментальных средств, которые существуют и применяются в современных разработках. Четырьмя из этих средств являются трассировка, трассировка снизу вверх, мгновенные дампы и постпечать. Опытные программисты, помнящие времена больших компьютеров, могут решить, что этот раздел посвящен выводу на печать или чтению системной диагностической информации, в том числе шестнадцатеричных системных дампов и сообщений редактора связей, для определения природы, места и исходного уровня причины системной ошибки. На самом деле это не так. Несмотря на сходные термины, мы определяем технологии, исходя из современных спецификаций и средств. Например, в программу, основанную на применении транзакций, проектировщики часто включают функцию трассировки транзакций, которая может избирательно включаться любым пользователем, выбирающим эту опцию в своей конфигурации. Эта опция создает файл трассировки с отображением входных и выходных данных транзакции и форматированные мгновенные дампы данных запроса. Естественно, испытатель должен тестировать систему без включения этой опции, поскольку для

пользователя по умолчанию выбирается именно этот режим. Но если кажется, что в приложении выполнения транзакций что-либо не в порядке, тестировщик должен проверить наличие опции трассировки транзакций, включить ее и сохранить журнальный файл, чтобы ошибку можно было повторить. Результаты трассировки должны быть приобщены к отчету об ошибке, что упростит воспроизведение условий ее возникновения.

При обнаружении ошибки недостаточно черкнуть в блокноте: "Не удалось загрузить файл", или включить подобную заметку в сообщение электронной почты, направленное автору кода. Большинство теоретиков тестирования призывают разделять задачи тестирования и разработки, чтобы тестировщик был *независимым* и отвечал только за обнаружение ошибок. Основной принцип быстрого тестирования заключается в объединении этих ролей в той мере, пока это способствует ускорению процессов обнаружения и устранения ошибок. Тестировщики могут поместить экранные снимки, например, в документ Microsoft Word, и снабдить их рабочими пометками или указателями на сценарии тестирования (см. рис. 10.4).

**Отчет об ошибке**

Время/дата: 9:40 пополудни 6/13/2001

Дата отчета: 9:50 пополудни 6/13/2001 по электронной почте в support<*a*>groove.com

Суть отказа:

В окне Groove Maintenance Update (Обновление программы Groove) я выбрал ссылку "Update Groove" ("Обновить Groove") и получил сообщение об ошибке, в котором говорилось, что не удалось найти URL-адрес. Это сообщение повторялось трижды, после чего, при четвертой попытке, я сделал следующую копию экрана.

В примере, приведенном на рис. 10.4, представлена неповторяющаяся ошибка. Через несколько дней после отправки отчета по этой ошибке тестировщик повторил этот тест. URL-адрес был найден, и загрузка выполнилась успешно. При этом использовалось то же клиентское программное обеспечение. Свидетельствует ли это о том, что никакой ошибки не было? Нет, в момент получения экранного снимка ошибка имела место. Отсутствие ошибки в клиентской программе означает лишь то, что ошибка была исправлена на сервере, после чего все стало работать правильно. В системах с архитектурой клиент/сервер ошибки распределяются по двум программным базам, и обслуживание таких систем отличается от обслуживания автономных программ. Эти различия будут темой одного из последующих разделов.

# Создание точек прерывания/правки

*Точка прерывания (breakpoint)* определяется как способ останова программного счетчика в какой-либо точке исходного кода. Отладчики, в случае их применения к программам на языке ассемблера, позволяют устанавливать точки прерываний для получения информации о состоянии памяти и регистров. Это дает возможность изменить данные в памяти или собрать информацию о распределении памяти, которая поможет отыскать способ исправления ошибки.

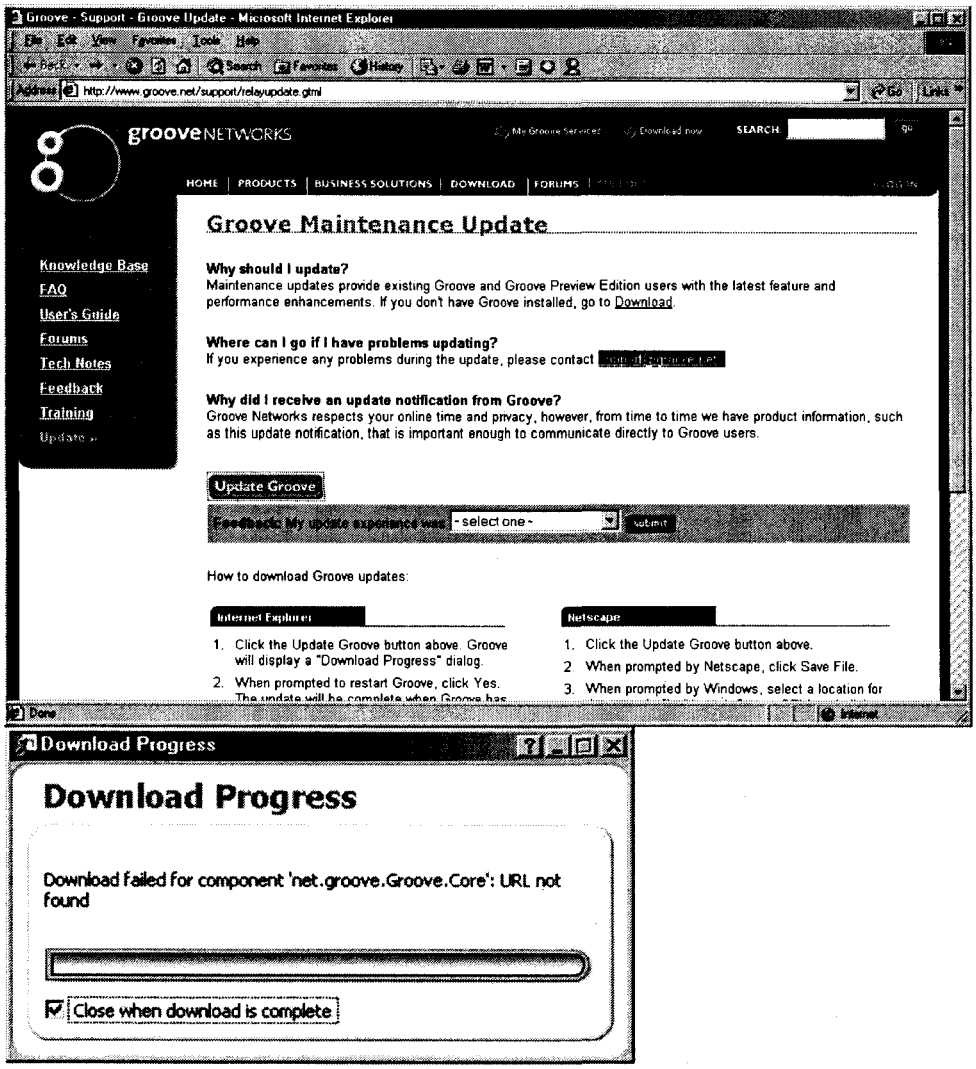



***Рис. 10.4. Экранные снимки, снабженные примечаниями.***

*Создание правок (patching)* — еще одна технология программирования на языке ассемблера, при которой определенная инструкция перехода переписывается, или правится, для перехода в область "заплаты" внутри адресного пространства программы. В этой области выполняется какой-то вспомогательный код, после чего осуществляется переход на инструкцию, следующую за исправленной. Обычно такой способ исправления ошибок применяется в больших программах, реализованных на языке ассемблера, например, Lotus 1-2-3.

Как механизм точек прерываний и правок можно применить в современном тестировании программ? Привлекательность этих технологий состоит в высокой скоро-

сти диагностирования и исправления ошибок, особенно тех, которые задерживают реализацию тестирования. Иногда тестировщик обнаруживает ошибку, которая препятствует выполнению дальнейшего тестирования в области данной ошибки. Такие ошибки, как определено в главе 5, называются блокирующими. Если подобную ошибку не исправить в кратчайшие сроки, это приведет к остановке всего процесса тестирования. Более чем для 95% ошибок, о которых сообщается команде разработки для их исправления, вполне приемлемо дождаться, пока группа контроля за внесением изменений определит приоритет и запланирует ресурсы для устранения каждой ошибки. Однако менее 5% блокирующих ошибок требуют немедленных действий, целью которых будет определение быстрого способа обхода заблокированных ветвей. Разработчик может предоставить временный способ исправления дефектного объекта с последующим частичным изменением связей, чтобы вернуть эту промежуточную/измененную версию персоналу тестирования. Это временное исправление будет повторно реализовано и протестировано в будущей полной версии, но пока группа тестирования может сохранить высокую эффективность обнаружения других ошибок в остальной части программы, начиная с точки блокирующей ошибки.

## Тестирование потока данных

Один из способов создания тестовых случаев для ошибок, связанных с потоками данных, которые проходят через невзаимодействующую с пользователем часть программы — это создание базовой копии файлов данных первоначального запуска программы. Затем выполняется сравнение результатов всех последующих запусков программы, при условии, что программа прогоняется с той же самой копией файлов данных. Наконец, в исходный код программы вставляются зонды, которые вносят в журнал уникальный номер зонда после выполнения каждого оператора ввода/вывода программы. Сравнивая номера зондов, записанные в журнал при каждом запуске программы, можно определить, остается ли поток данных через зонды неизменным для каждого теста, который выполняется при использовании одной и той же базовой копии файлов данных первоначального запуска. Затем этот тестовый случай может применяться для проверки неизменности потока данных и в новой версии.

Более распространенным способом тестирования потоков данных является подход, который используется для тестирования приложений, ориентированных на транзакции, например, систем ввода заказов, систем обработки заказов, технологических систем, систем выписки счетов, поставки и возврата продукции. Данные, полученные в результате каждой тестовой транзакции, могут отслеживаться в каждом из этих процессов и подтверждаться промежуточными результатами тестирования, выполняемого между процессами.

В случае тестирования изменений существующих систем планирование и выполнение тестирования потоков данных не связано с особыми сложностями, поскольку снимки системы можно сделать до и после того, как каждый процесс впервые используется существующей системой. Впоследствии, после внесения изменений в новую версию и ввода в нее тех же самых данных, новые результаты можно сравнить (иногда автоматически) с предыдущими результатами тестирования существующей системы.

Еще одним примером блокирующей ошибки, которая также влияет на поток данных, может послужить оператор:

```
IF (I > J) THEN B(I) = A(J) ELSE B(I) = 0 ENDIF.
```

Если символ операции "больше чем" (>) указан по ошибке и вместо него должен присутствовать символ операции "меньше или равно" (<=), обнулится неверная часть массива B. Все вычисления, следующие за этим оператором IF будут иметь дело с этими обнуленными значениями B, в то время как они должны выполняться с ненулевыми значениями, скопированными из массива A. Это означает, что все результаты теста, зависящие от B, окажутся неверными. Поэтому тестировщик не сможет определить, содержит ли ошибки остальная часть программы (та, что следует за показанным оператором IF). Тестировщик мог бы проследить поток данных для всех значений I и J или их некоторого поднабора. Имея информацию о потоке данных через несколько процессов, специалист по тестированию может извлекать промежуточные результаты, тем самым ускоряя процесс выявления и последующего воссоздания ошибок.

## Тестирование на предмет утечек памяти

Утечка памяти — это тип ошибки, которая приводит к тому, что приложение постепенно заполняет всю виртуальную память, управляемую операционной системой. Множество современных языков организуют в виртуальной памяти стек и кучу, обеспечивая более эффективное и удобное управление временной областью памяти. Утечки памяти имеют несколько других названий, в числе которых паразитное распределение памяти, разрушение стека, разрушение памяти, квазипереполнение. Основная причина возникновения ошибки утечки памяти — невозможность освобождения виртуальной памяти, которая динамически запрашивается приложением. Не существует никаких внешних средств устранения утечки памяти, за исключением избежания рабочей области приложения, вызывающей утечку памяти. Большинство сотрудников служб технической поддержки просто рекомендуют прервать выполнение приложения или перезагрузить операционную систему. На рис. 10.5 показан пример сообщения группы технической поддержки компании Microsoft относительно утечки памяти в поступившей в продажу версии операционной системы Windows 95.

Компания Microsoft поместила средства обхода и исправления этих ошибок утечки памяти на соответствующие Web-сайты.

Как правило, утечка памяти происходит в небольшой функции, которая правильно написана для своей *удачной ветви,* но неправильно — для *неудачной ветви.* Если выделение виртуальной памяти выполняется во время запуска функции, но ввод данных пользователем приводит к прерыванию программы, обработчик ошибок для данного прерывания может быть не запрограммирован для освобождения виртуальной памяти. Независимо от того, велик или мал объем выделяемой памяти, повторяющийся характер функции, в конечном счете, обусловит уменьшение объема доступной виртуальной памяти, и программа больше не сможет функционировать. Особенно уязвимы в этом отношении приложения, которые выполняются круглосуточно в течение всех семи дней недели. Даже самые малые утечки памяти со временем приведут к разрушению программы или системы.

**УТЕЧКА ПАМЯТИ В ЯДРЕ WINDOWS 95 ПРИ ИСПОЛЬЗОВАНИИ СОКЕТОВ WINDOWS**

**СИМПТОМЫ**

При запуске программы, которая использует сокеты Windows, в среде Windows 95 со временем может происходить увеличение объема памяти, используемой операционной системой, особенно если программа открывает и закрывает множество каналов.

**ПРИЧИНА**

Ядро Windows 95 (Kernel32.dll) содержит ошибку, которая препятствует корректному освобождению определенных небольших структур данных, связанных с процессами сокетов Windows и выделенными каналами. Со временем эти небольшие утечки памяти могут приводить к существенному уменьшению объема доступной памяти.

Имейте в виду, что ресурсы, связанные с программой, можно освободить, закрыв программу. После выхода из программы и перезапуска Windows 95 память освобождается.

*Рис. 10.5. Поступившая в продажу версия Windows 95 с утечками памяти операционной системы.*

Утечки памяти особенно трудно выявить в объектно-ориентированных исходных кодах. Тем не менее, при помощи трассировки и отслеживания *гистограммы объектов (Object Histogram),* разработчик может выяснить, какой объект выделяет и освобождает память и какие вызовы присутствуют между этими вызовами диспетчера памяти. Перед загрузкой и запуском сомнительной Java-программы саму среду следует запустить с параметром *-tracepopulation.* Как только объект, вызывающий утечку памяти, локализован, не забудьте воспользоваться технологией статического тестирования для ускорения выяснения основной причины. Еще важнее разобраться в коде и любых возможных прерываниях в *удачной ветви,* которые могут помешать объекту освобождать память. В C++ существует объект *debug_malloc,* который может оказаться очень полезным для трассировки утечки памяти. Хотя утечки памяти не носят перемежающийся характер, они быстрее проявляются в бездисковых системах с небольшим объемом памяти, нежели в конфигурациях с большим объемом оперативной памяти и дискового пространства, перезагрузка которых выполняется редко (например, в сервер ных приложениях).

Исходя из концепции быстрого тестирования, можно дать следующий совет. Как только разработчик обнаружил и устранил утечку памяти, он должен проверить новый код на предмет наличия в этой области других ошибок, могущих привести к утечкам памяти. Достаточно часто, столкнувшись с первым проявлением ошибки, разработчик считает ее единственной, упуская из виду другие ситуации, когда этот же код может не освобождать динамически распределяемую память. Этот совет противоречит также призывам большинства инструкторов по тестированию и менеджеров конфигурации заниматься устранением только той проблемы, которая указана в отчете об ошибке. Если вы хотите следовать советам по быстрому тестированию, иногда приходится отказываться от трафаретного подхода и просто заниматься повтор ным проектированием и реализацией приложения.

# Тестирование интерфейса "человек-компьютер"

Интерфейс "человек-компьютер" (human-computer interface, HCI) в современных компьютерах может охватывать широкое множество различных периферийных устройств, таких как клавиатура, клавишная панель, мышь, световое перо, монитор с сенсорным экраном, речевой интерфейс, сканер и многие другие. Кроме того, в случае использования больших систем управления производством, работающих на основе больших вычислительных машин, для работы с программой может требоваться большое количество операторов, каждый из которых имеет собственный персональный компьютер, действующий как терминал большой вычислительной машины. Планирование тестирования исключительно важно для координации тестирования интерфейса "человек-компьютер", поскольку, возможно, это — единственное тестирование, при котором продукт рассматривается полностью с точки зрения пользователя. Через этот интерфейс доступны для тестирования многие функциональные требования пользователя, а именно это и есть основная цель системных испытаний.

При использовании интерфейса "человек-компьютер" для выполнения тестирования с точки зрения пользователя следует учитывать одно важное обстоятельство. Если команда тестирования начинает тестирование с этапа системных испытаний и при этом интенсивно использует интерфейс "человек-компьютер", диапазон тестирования будет очень ограничен. Тестирование интерфейса "человек-компьютер" — лишь один из множества способов выполнения быстрого тестирования с целью обнаружения ошибок. Фактически, как правило, процесс тестирования интерфейса "человек-компьютер" оказывается не слишком эффективным при поиске даже наиболее общих ошибок, поскольку этот процесс больше ориентирован на проверку маршрута прохождения потока данных по продукту, а этот маршрут уже подвергался интенсивному поблочному тестированию, выполняемому по тщательно разработанным сценариям.

При организации тестирования интерфейса "человек-компьютер" всегда полезно, чтобы с членами команды тестирования сотрудничал администратор баз данных и несколько конечных пользователей. Они могут разрабатывать тестовые случаи, сценарии тестирования и методику обработки результатов тестирования для проверки правильности отображения входных данных на экране. Далее такая информация сохраняется в соответствующих записях базы данных, обрабатывается и участвует при составлении отчетов. Администратор баз данных и пользователи могут также создавать тестовые случаи, сценарии тестирования и методику обработки результатов тестирования для проверки правильности форматирования выходных данных и их передачи на соответствующий носитель или в коммуникационные буферы.

Конечные пользователи могут оказать существенную помощь, сотрудничая с группой тестирования при разработке объективного приемочного теста. В больших организациях существует должность бизнес-аналитика. Как правило, бизнес-аналитик выполняет роль своего рода конечного пользователя, который оплачивает работы из фонда проекта и организует тестирование интерфейса "человек-компьютер", призванное определить приемлемость данной версии программного обеспечения. Приемочные тесты, выполняемые конечными пользователями, как правило, будут выполняться в соответствии с определенным сценарием и будут исследовать только

удачные ветви программы. Тестовые случаи, сценарии тестов и методики обработки результатов тестирования, разработанные конечными пользователями, могут быть не такими строгими, как применяемые в ходе выполнения проекта. Но, несмотря на это, найденные ими ошибки документируются, исправляются и повторно тестируются командой тестирования. Затем внесенные изменения, ориентированные на исправление ошибок, снова тестируются конечными пользователями в ходе дополнительных приемочных испытаний. С другой стороны, для обеспечения высокой надежности версии программы и того, чтобы тестовые случаи, сценарии тестов и методики обработки результатов тестирования отражали основные случаи использования, тестирование, предпринимаемое бизнес-аналитиком, должно охватывать как удачные, так и неудачные ветви.

## Тестирование нагрузочной эффективности

Эффективный с точки зрения затрат способ расширения приложений заключается в использовании распределенных вычислений. Существует два основных компонента сетевой архитектуры: сервер и клиент. Сеть может иметь тысячи серверов и десятки тысяч настольных персональных компьютеров, переносных компьютеров, беспроводных и карманных клиентов, каждый из которых подключается к серверам по сети. Системы клиент/сервер являются одними из наиболее широко распространенных, и огромная доля выполняемого в наши дни тестирования связана со средами типа клиент/сервер/сеть. Тестирование требований безопасности и доступа относится скорее к тестированию системы, чем к тестированию приложения. Для баз данных могут потребоваться процессы создания экземпляров, поскольку приложение, выполняющееся в одном городе/стране, при запросе базы данных должно получать те же результаты, что и это же приложение, одновременно выполняющееся в другом городе/стране.

Тестирование приложений типа клиент/сервер требует глубокого понимания архитектуры программного обеспечения, в том числе серверов хранилищ данных, характеристик путей передачи информации и серверов балансирования нагрузки. Часто для успешности тестирования первостепенную важность имеет определение тестовых данных. В этом разделе приведены лишь несколько примеров, но соответствующие аналогии можно найти и в других приложениях.

При тестировании приложений, в которых интенсивно используется обмен данными, должны применяться два подхода: использование тестовых шаблонов и рандомизированных потоков данных. Если основной целью тестирования является проверка надежности коммуникационного пути, то можно задействовать тестирование **с** использованием шаблонов. Выявить искажение в повторяющемся графическом шаблоне значительно легче, чем просматривать распечатку со значениями данных для выяснения того, присутствует ли в них какая-то ошибка. Инженеры по электронике аналогичным образом используют осциллографы, просматривая осциллограммы сигналов тестовых частот для проверки правильности функционирования соединения или процесса. Специалисты по тестированию должны использовать эту технологию в ходе процесса самотестирования коммуникационного пути во время начального запуска системы. Как только коммуникационный путь проходит процесс самотестирования при начальном запуске, по нему можно передать заранее известный набор

данных транзакции, что позволит проверить правильность выполнения транзакции и производительность процессора транзакций. Выходные данные, сохраненные в выходных записях или журнальных файлах, можно автоматически сравнить с ожидаемыми результатами. Как правило, такие тесты разрабатываются в форме регрессивных тестов для систем выполнения транзакций, которые постоянно обновляются из-за добавления в производственную среду новых транзакций.

Приложения клиент/сервер поддерживают возможность применения тестовой конфигурации, в которой программируются и загружаются сервер эмуляции высокого темпа передачи данных и тестовый эмулятор хранилища данных. При такой конфигурации можно проводить испытания банка производственных серверов, которые подключаются к системе через сервер балансирования нагрузки, как показано на рис. 10.6. Сервер имитации высокого темпа прогоняет данные транзакций по альтернативному пути к серверу балансирования нагрузки, который, в свою очередь, прогоняет эти данные через банк производственных серверов со скоростью, превышающей обычные рабочие скорости передачи данных. Этот тест может выявить узкие места с точки зрения производительности. При наличии ветви обратной связи от тестового эмулятора хранилища данных к серверу имитации высокого темпа передачи, возможно автоматическое сравнение выходных данных транзакций производственного сервера с ожидаемыми выходными данными тестов. Эта замкнутая система может использоваться для проведения лабораторных испытаний новых версий перед их погружением в производственный процесс. Данная технология непосредственно применима и к большинству Web-приложений.

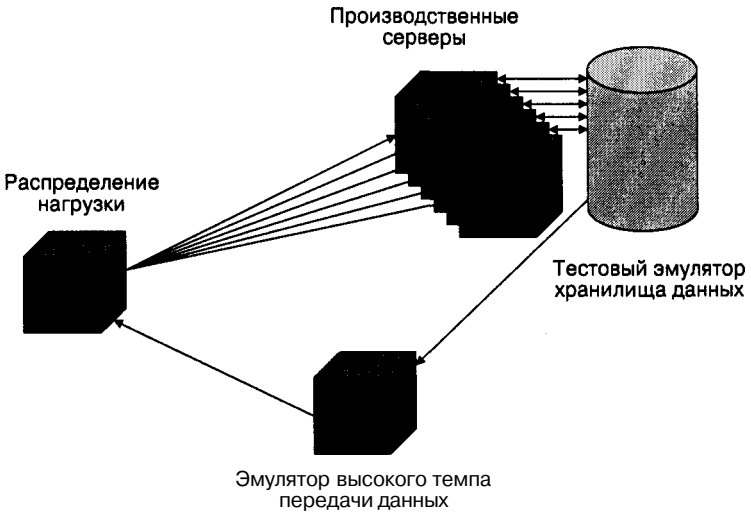



*Рис. 10.6. Сервер имитации высокого темпа передачи данных и тестовый эмулятор хранилища данных испытывают производственное приложение с архитектурой клиент/сервер.*

### ПРИМЕР: СОЗДАНИЕ ТЕСТОВЫХ ДАННЫХ (ГЭРИ КОББ)

Когда я работал в составе группы разработки программного обеспечения (Software Engineering Process Group, SEPG), ко мне подошел один из инженеров по тестированию и спросил, что мне известно о силах Кориолиса. Я сообщил ему, что, насколько мне известно, это сила, возникающая в результате вращения Земли, которая оказывает влияние на любое движущееся тело на ее поверхности или в толще земной коры, вызывая отклонение от курса. Мы говорили о том, что силы Кориолиса вызывают отклонение судов и самолетов вправо от курса в северном полушарии и влево — в южном. Эта сила оказывает также влияние на морские и воздушные течения.

Инженер по тестированию спросил меня, не согласился бы я помочь в создании входных данных для тестовых случаев, которые будут использоваться во время тестирования недавно разработанной бортовой системы прогнозирования погоды. Я согласился и решил при разработке данных воспользоваться электронными таблицами. Одной из причин такого выбора послужило то, что в электронных таблицах все числовые вычисления выполняются в 64-разрядном модуле арифметических операций с плавающей точкой. Все тестовые данные, которые были нужны инженеру по тестированию, показаны на рис. 10.

Та-
юп-4
тес-4

```
G E O S T R O P I C W I N D C O M P U T A T I O N
D U E T O C O R I O L I S

      CONSTANTS:

      EARTH RADIUS  (3437.911nm*1852.44m/nm)  6368523.85284 m
      EARTH'S GRAVITY                         9.8 m/sec**2
      EARTH'S ANGULAR VEL.  ("Omega")         7.2920E-05 per second

      DEFINE AREA OF INTEREST  (AOI):

      CENTER OF AOI                           C. LA             C.LO
      CENTER  (DEGREES)                       30.199998         -97.800000
      CORIOLIS FORCE AT CENTER OF AOI         7.33 60E-05 per second
      AOI WIDTH  (3200nm*1852.44m/nm)         5927808 m
      M=MESH=l/64                             5788.875 m resolution
      GRID SIZE                               1024 X 1024

DL = DELTA LONGITUDE  (DEGREES)  =  (AOI WIDTH)/((PI/180)*(EARTH RADIUS))
      DL                                      53.330785596795 degrees

      AOI POINTS:                             x.LA x.LO
      CENTER(C)  (AUSTIN)                     30.199998         -97.800000
      EAST(R.C.X)                             30.199998         -124.465392
      NORTH(U.C.X)                            56.865391         -97.800000
      WEST(L.C.X)                             30.199998         -71.134607
      SOUTH  (L.C.X)                          3.534605          -97.800000

                                              MX                MY
      CENTER(C)  (AUSTIN)                     0                 0.553340844437
      EAST(R.C.X)                             -0.46539890067    0.553340844437
      NORTH(U.C X)                            0                 1.212368963348
      WEST(L.C.X)                             0.4653989006696   0.553340844437
      SOUTH  (L.C.X)                          0                 0.061729665232

ANGLE SUBTENDED BY ARC 100 NM LONG ON THE EARTH SURFACE:
```

```
ЮОпт* (1852. 44m/nm) / (earth radius (m) ) * (180/PI)
      Angle(deg)                              1.6665870498998  degrees

      SCALE FACTOR(SF)= MD/SD = ABS(U.C.MY-L.C.MY)/SY.MAX
      SF =                                    0.00112477  (earth radii)

      AOI CORNERS:                            x.LA              x.LO
      U.LE.X                                  29.36670447505    -96.9667059751
      U.RI.X                                  31.03329152495    -98.6332930249
      L.LE . X                                9.36670447505     -96.9667059751
      L.RI.X                                  28.5334109501     -98.6332930249

                                              MX                MY
      U.LE.X                                  -0.575319649058   1.212368963348
      U.RI.X                                  0.5753196490579   1.212368963348
      L.LE.X                                  -0.575319649058   0.061729665232
      L.RI.X                                  0.5753196490579   0.061729665232

BOX CENTERS  (DEGREES)                        x.LA              x.LO
      ENTER BOX                               33.3166666667     113.7500000000
      EAST BOX                                33.3166666667     111.6166666667
      NORTH BOX                               35.0833333333     113.7500000000
      WEST BOX                                33.3166666667     115.8166666667
      SOUTH BOX                               31.5833333333     113.7500000000

                                              MX                MY
      CENTER BOX                                                0.617329479414
      EAST BOX                                -0.037233690709   0.617329479414
      NORTH BOX                                                 0.654613029744
      WEST BOX                                                  0.0360701378745
      SOUTH BOX                                                 0.581477015532

dX=ABS(DELTA LON.)*(EARTH RADIUS  IN m)*COS(LAT.IN RAD.)*(PI/180)
East-West  dX                                 390111.37991131 m

dY=ABS(DELTA LAT.)*((EARTH RADIUS  IN m)*PI/180)
North-South dY                                389030.98403348 m

Test Case 1. Assume following heights of the 300 Mb constant pressure field:
      CENTER BOX                              9000
      EAST BOX                                9100
      NORTH BOX                               9100
      WEST BOX                                8900
      SOUTH BOX                               8900

      GEOSTROPHIC WIND VECTOR COMPONENTS  (Ug directed East,  Vg directed North)
      Ug = -(GRAVITY/(2*OMEGA*SIN(LAT.(RAD)))*(PARTIAL OF H WRT Y)  =
      Ug  (in m/sec)                          -62.72030217113   -51.39

      Vg =  (GRAVITY/(2*OMEGA*SIN(LAT.(RAD)))*(PARTIAL OF H WRT X)
      Vg  (in m/sec)                          62.89448561333    51.69
```

*Рис. 10.7. Тестовые данные для учета влияния сил Кориолиса.*

Использование таких электронных таблиц при разработке данных для тестовых случаев дает явные преимущества. Как только подтверждается достоверность этих вычислений, они могут повторно использоваться для вычисления тестовых результатов в других интересующих областях.

# Тестирование конфигурации платформы

Вариации в конфигурации существенно осложняют тестирование. Одним из таких осложняющих факторов могут рассматриваться периферийные устройства. Каждое периферийное устройство поставляется с драйвером фирмы-изготовителя, который выбирается из набора драйверов для различных операционных систем. Именно поэтому персональные компьютеры являются персональными. Из-за сложности и высокой стоимости операционных систем и периферийных устройств тестирование приложения для всех возможных конфигураций персональных компьютеров оказывается слишком дорогостоящим. Тестировщик, занимающийся быстрым тестированием, должен относится к нему, как к тестированию ветвей, где полный охват не возможен, и применять тестовые случаи только к преобладающим комбинациям операционных систем и периферийных устройств.

Для того чтобы упростить управление набором тестовых платформ, можно воспользоваться технологией разработки матрицы тестовых конфигураций, которая ускоряет планирование и реализацию процесса тестирования и способствует организации процесса составления отчета по результатам. Предлагаемый формат матрицы тестовых конфигураций показан в таблице 10.3. Элементы, приведенные в столбцах H1 и H2, должны иметь подробные спецификации файла, описывающего номера версий программно-аппаратного обеспечения и положений переключателей для любого реконфигурируемого оборудования.

Если матрица платформ большая и сложная, она может быть базой данных, обеспечивающей датирование транзакций и регистрацию в журналах для реализации управления конфигурациями. Для отражения изменений в аппаратном обеспечении можно с заданной периодичностью выполнять проверки этой матрицы, призванные дать ответ на вопрос: выполняется ли тестирование оборудования на самых современных платформах, занимающих такое же место на рынке, как и тестируемый программный продукт? Эта матрица или база данных тестовых конфигураций должна совместно использоваться всеми партнерами, несущими ответственность за тестирование конечного программного продукта.

Обычно при поставке программного продукта на общий потребительский рынок принято перечислять основные требования к платформе. Пример требований к платформе, указываемых при поставке на потребительский рынок, приведен в таблице 10.4. Информация из этой таблицы не совпадает с данными, приведенными в матрице тестовых конфигураций. Например, при указании операционной системы для тестовой конфигурации необходимо упоминать номер версии и уровень сервисного пакета. Кроме того, жесткие диски должны указываться вместе с названием фирмы-изготовителя, версии программно-аппаратного обеспечения, версии драйвера и любых других параметров. Тестировщикам придется разработать отрицательные тесты для нескольких конфигураций с отсутствующими или неподходящими компонентами. Это даст возможность задокументировать то, как приложение обрабатывает каждую из неподдерживаемых конфигураций.

**Таблица 10.3. Матрица тестовых конфигураций**

| ID | Компоненты | H1 | H2 | Прочее (дополнительные столбцы) |
|---|---|---|---|---|
| 1 | Аппаратное обеспечение | PC-100, Pentium III | PC-100, Pentium III | |
| 2 | Операционная система | Linux 7.0 Windows | Win98SE, SP-2 | |
| 3 | Клавиатура | Happy Hacking Lite | Standard Querty | |
| 4 | Клавишная панель | | | |
| 5 | Мышь | IntelliPoint | Трехкнопочная мышь | |
| 6 | Перо | | | |
| 7 | Сенсорный экран | HP | | |
| 8 | Речевой интерфейс | | Via | |
| 9 | Сканер | Plustek | | |
| 10 | Модем | Psion V.90 | Xircom 56 бод | |
| 11 | База данных | MS Access | Oracle 7.0 | |
| 12 | Объект тестирования | Build 2.4.3 | Build 2.4.4 | |
| 13 | Добавьте дополнительные элементы, вставив дополнительные строки | | | |

**Таблица 10.4. Основные требования к платформе**

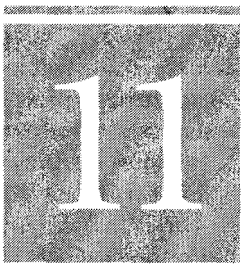
| ID | Системные требования |
|---|---|
| 1 | Рекомендуется использовать ПК с процессором Pentium 166 (или более производительным) |
| 2 | Microsoft Windows 95, 98 или NT |
| 3 | 30 Мб свободного дискового пространства |
| 4 | Не менее 32 Мб свободного пространства в ОЗУ |
| 5 | Клавиатура и устройство указания |
| 6 | Дисковод CD-ROM (рекомендуется использование 4-скоростного или более производительного дисковода) |
| 7 | Видеоплата, обеспечивающая цветовое разрешение 16 бит или TrueColor |
| 8 | Звуковая плата, совместимая с Microsoft Windows |
| 9 | Драйверы MCI и видеодрайверы |
| 10 | Принтер (не обязательно) |
| 11 | Доступ к Internet (не обязательно) |

## Резюме

**В** дополнение к технологиям и советам по статическому тестированию, упомянутым в главе 9, в главе 10 освещены дополняющие их технологии и советы по динамическому тестированию. Каждый специалист по тестированию должен поэкспериментировать с этими технологиями и оценить, насколько каждая из них применима для конкретных целей тестирования. Неплохо описать собственный опыт по использованию новой технологии и добавить этот материал к рассмотренному в данной главе. В этой главе было показано, что несмотря на то, что много участников тратят массу времени на тестирование продукта с точки зрения пользователя и с помощью графического интерфейса, существует ряд очень важных тестовых случаев, которые должны быть разработаны с целью оценки аспектов продукта, отличных от графического интерфейса. Расширяемость, устойчивость к ошибкам, пригодность к высокому темпу передачи данных, возможность использования в средах поврежденных платформ и точность результатов считаются наиболее важными областями выявления ошибок в конечном продукте. Параллельное выполнение тестовых случаев требует координирования и, в ряде случаев, участия назначенного координатора тестирования. Применение технологий статического тестирования к результатам тестирования для тщательной их проверки на предмет возможных ошибок представляется разумной тратой времени на последнем этапе разработки.

Памятуя об этом, приходится лишь удивляться, когда известие в какой-то организации о том, что продукт прошел приемочные испытания, вызывает восторг, а зачастую и удивление. Так не должно быть. Каждый член команды разработки должен нести достаточную ответственность за выполнение тестирования, чтобы иметь уверенность в том, что конечный продукт полностью соответствует требованиям пользователей, и в будущем будет способствовать процветанию организации.

# Разработка и использование показателей тестирования: моделирование и прогнозирование ошибок




**Темы, рассматриваемые в главе:**

- Определение показателей и данных измерений
- Использование стандартных показателей для внесения усовершенствований
- Показатели тестирования
- Проектно-ориентированная модель ошибок
- Программа оценки ошибок программного обеспечения (SWEEP)
- Резюме

При планировании нового проекта кто-то обязательно задает вопрос: "А где какие-либо фактические данные наших предшествующих проектов?" Может возникать и вопрос: "Какая связь между планом и данными измерений?" План полезен только в том случае, если он расходится с действительностью не более чем на +/-20%. Данные измерений полезны, только если они применимы к проекту, условия которого аналогичны условиям предыдущего проекта, когда эти данные были получены.

Представьте себе, что вы являетесь владельцем небольшой океанской яхты и собираетесь плыть из Атлантик-Сити, Нью-Джерси, на Багамы. Вам пришлось бы подобрать карты атлантического побережья с отображением морских путей, глубин, портов и маяков. Вам потребовалось бы выбрать маршрут и подсчитать время путешествия с учетом быстроходности яхты и любых ограничений скорости, указанных в навигационных картах. При планировании необходимо выполнить ряд вычислений с учетом скорости расходования различных припасов. Вам следовало бы решить, сколько человек планируется взять с собой в путешествие. Промежуточные результаты, подобные количеству дней путешествия, определяют количество порций завтраков, обедов или ужинов, которые потребуются, чтобы прокормить присутствующих

на борту яхты людей. В соответствии с этим планом вы запасетесь соответствующим количеством топлива, питьевой воды, пищи и других припасов. Эта информация находит свое отражение в навигационном и продовольственном планах, в судовой декларации и карте маршрута.

В море капитан яхты будет сверяться с навигационным планом, записывая такие данные измерений, как скорость и текущий курс. На основе этих данных он вычисляет время, когда следует начинать высматривать конкретный порт или маяк. Назначение этого набора фактических данных измерений состоит в привязке навигационного плана к известным пунктам маршрута. Отслеживая эти данные измерений и сравнивая их с контрольными пунктами, капитан может ответить на такие наиболее типичные вопросы, как "Мы еще не сбились с курса?" или "Не выбились ли мы из графика?".

Применительно к проектам разработки программного обеспечения процесс планирования заключается в объединении выбранного жизненного цикла разработки и его этапов (навигационные карты и маршрут), набора функциональных требований (протяженность путешествия в милях), нормативов по количеству строк кода, которые должны быть созданы за час работы (предварительно определенная скорость яхты, выраженная в милях/ч), графика загрузки персонала по месяцам (маршрут) и промежуточных сроков выполнения проекта (маяки и порты).

В ходе разработки руководитель проекта фиксирует фактические данные, например, полученные в течение текущей недели данные по количеству вариантов использования, количеству строк кода, количеству обнаруженных и устраненных ошибок и процентному отношению выполненных тестовых случаев к общему их количеству. Кроме того, руководитель разработки программного обеспечения просматривает план проекта и его календарный график, чтобы иметь возможность ответить на типичный вопрос: "Не срываются ли сроки выполнения проекта?"

В главе 11 приведены определения показателей и описаны способы сбора и анализа данных измерений. Глава 12 посвящена вопросам определения необходимого штата и календарного графика выполнения работ для стандартного жизненного цикла с использованием стандартной модели оценки затрат. Концепция быстрого тестирования предполагает применение программных показателей, разработанных на основе сбора и использования данных измерений, полученных в ходе выполнения предыдущих проектов быстрого тестирования. В результате достигается постоянное совершенствование процессов, повышение качества, снижение затрат и сокращение плановых сроков поставки продукта.

## Определение показателей и данных измерений

Программный показатель — это стандартный способ измерения какой-либо характеристики процесса разработки программного обеспечения. Данные измерений — это числовые данные, собранные и зафиксированные в единицах измерения, определенных для показателя. Например, если показателем является количество скрытых ошибок на тысячу строк кода (K lines of code, KLOC), то набор данных измерений мог бы выглядеть следующим образом: {Программа A: 2,6 ошибок/KLOC; программа B: 12,1 ошибок/KLOC; программа C: 5,8 ошибок/KLOC}.

Основные преимущества применения программных показателей заключаются в следующем:

- В возможности использования исходных сравнительных данных совместно с другими проектами разработки.
- В возможности сбора данных о состоянии, выражаемых в одних и тех же единицах, во всей организации для определения успешности выполнения стоящей задачи.
- В возможности выполнить прогнозирование сравнительных трудозатрат.
- В возможности оценить трудоемкость или календарный план, исходя из имеющегося опыта разработок.
- В возможности во время формальных пересмотров составлять отчеты по данным, характеризующим тенденции процесса.

Данные измерений программы должны быть получены или преобразованы в числовые единицы измерения, определенные для программного показателя. Эти данные должны характеризовать атрибуты программного процесса, его результаты или календарный план или ресурсы проекта разработки. Несколько примеров программных показателей, разделенных по стандартным промышленным категориям, приведено в таблице 11.1.

Прежде всего, следует уяснить, что существуют тысячи возможных программных показателей. При измерении большинства из них необходима методичность и большие затраты времени, и для одной конкретной группы разработки они могут оказаться экономически неэффективными. В общем случае выбор показателей из списка, подобного приведенному в таблице 11.1, может быть сродни хождению по минному полю, если только не подходить к процессу определения и использования показателей очень прагматично. В этом смысле не являются исключением и показатели тестирования. Ниже приведен практический пример, представляющий точку зрения одного из авторов и иллюстрирующий применяемый им подход к определению показателей, который удовлетворяет потребности организации и при этом не становится обузой ни для одного из проектов внутри организации. В данном случае термин "организация" употребляется в отношении нескольких проектов разработки программного обеспечения, выполняемых под одним и тем же руководством (например, в каком-нибудь подразделении более крупной компании).

**Таблица 11.1. Примеры категорий программных показателей**

| Категория | Программный показатель |
|---|---|
| Размер | Общее количество написанных строк кода |
| | Количество строк комментариев |
| | Общее количество объявлений |
| | Общее количество пустых строк |
| | Количество точек вызова функций |
| | Количество объектов |
| Производительность | Количество рабочих часов, затраченных на разработку проекта |
| | Количество рабочих часов, затраченных на каждый объект |
| | Количество изменений каждого объекта |
| Удобство сопровождения | Количество параметров, передаваемых каждому объекту |
| | Количество методов, приходящихся на один объект |
| | Количество объектов, использующих тот или иной метод |
| | Количество точек принятия решений в каждом объекте |
| | Сложность управляющей логики каждого объекта |
| | Количество строк кода в каждом объекте |
| | Количество строк комментариев в каждом объекте |
| | Количество объявлений данных в каждом объекте |
| | Количество прямых ветвей в каждом объекте |
| | Количество операторов ввода и вывода в каждом объекте |
| Трассировка ошибок | Уровень серьезности каждой ошибки |
| | Местоположение каждой ошибки |
| | Способ исправления каждой ошибки |
| | Лица, несущие ответственность за каждую из ошибок |
| | Количество строк, на которые ошибка оказала влияние |
| | Время, потребовавшееся для отыскания ошибки |
| | Время, потребовавшееся для устранения ошибки |
| | Количество предпринятых попыток исправления каждой ошибки |
| | Количество новых ошибок, допущенных в результате исправления каждой из ошибок |
| Качество | Общее количество ошибок |
| | Количество ошибок в каждом объекте |
| | Среднее количество ошибок на тысячу строк кода |
| | Среднее время наработки на отказ |
| | Количество ошибок, обнаруженных компилятором |

### ПРИМЕР: ПРОГРАММА ОПРЕДЕЛЕНИЯ ПОКАЗАТЕЛЕЙ (ГЭРИ КОББ)

В течение примерно шести лет автору довелось выступать в роли координатора разработки программных показателей в организации, занимающейся созданием программного обеспечения, в которой трудились около 450 разработчиков. Будучи членом группы координации процесса разработки программного обеспечения (Software Engineering Process Group — SEPG), функции которой определены в модели развития функциональных возможностей (Capability Maturity Model, CMM V1.1) института технологий программного обеспечения SEI, мне пришлось изучить более 400 программных показателей, предлагаемых SEI. На основе этого исследования я пришел к выводу, что в конкретной организации следует стандартизировать лишь небольшое количество показателей. Основная масса показателей должна разрабатываться, отбираться, анализироваться и использоваться каждой группой в зависимости от этапа разработки.

При определении программы создания показателей для нескольких независимых проектов разработки программного обеспечения я установил ряд общих стандартов показателей, отчеты о которых должны были поступать из всех проектов и анализироваться в SEPG. Но одновременно мною был создан процесс, который позволял в каждом проекте определять и использовать собственные показатели, которые не нужно было включать в отчет или переносить в другие проекты. Стандартный набор из четырех программных показателей, который был предложен, утвержден и принят для использования во всех проектах разработки программного обеспечения, описан в таблице 11.2.

**Таблица 11.2. Общий для всей организации набор программных показателей**

| Название показателя | Определение |
|---|---|
| SIZE (РАЗМЕР) | Количество тысяч эквивалентных строк кода (KESLOC) |
| EFFORT (ЗАГРУЗКА) | Количество неадминистративных работников, занятых в разработке проекта в течение месяца |
| SHEDULE (КАЛЕНДАРНЫЙ ПЛАН) | Календарные сроки (по месяцам) выполнения отдельных этапов жизненного цикла разработки программного обеспечения |
| QUALITY (КАЧЕСТВО) | Количества ошибок на тысячу эквивалентных строк кода |

Ожидалось, что фактические данные по каждому из этих показателей будут сообщаться в ходе формального пересмотра, проводимого на каждом из основных промежуточных этапов жизненного цикла разработки. Эти основные промежуточные этапы и связанные с ними формальные пересмотры соответствуют основным этапам каскадной модели жизненного цикла. В их число входят эскизное проектирование (ЭП), рабочее проектирование (РП), тестирование кода и модулей (ТКМ) и комплексные и системные испытания (КИ). В качестве примера на рис. 11.1 приводится диаграмма показателя размера программы, которую можно представить во время формального пересмотра на этапе ТКМ. Часто большая система допускает разложение на основные подсистемы, представленные на диаграмме отдельными слоями. В приведенном примере размер программы в значительной степени изменяется от одного этапа к другому. Не вдаваясь в подробности, можно утверждать лишь следующее. На этапе разработки требований был определен приемлемый размер проекта, но на этапе ЭП предполагалось, что свыше 100000 строк кода будут использоваться повторно. Впоследствии, по завершении этапа ЭП выяснилось, что фактически код повторно использоваться не может. В результате оценки размеров модулей были пересмотрены, что и нашло свое отражение в диаграмме, начиная с этапа РП.

На рис. 11.2 приведена диаграмма фактических данных по занятости персонала, на которой показана фактическая загрузка персонала в течение первых четырех месяцев (т.е. 16 недель) и предполагаемая загрузка до окончания проекта (начиная с 16-ой недели). В случае выполнения масштабных проектов данные измерений загрузки персонала разбиваются по этапам и/или характеру выполняемых работ.

В данном случае такими этапами являются: эскизный проект (ЭП), рабочий проект (РП), программирование (П), тестирование модулей (ТМ), системные испытания (СИ), прочие работы (Пр) и вспомогательные работы (ВР). В процессе анализа этой диаграммы *за*грузки персонала у руководителя разработки должен был бы возникнуть естественный вопрос: "Что на четвертом месяце привело к уменьшению количества занятых в проекте людей?" Судя по представленным данным, в течение 12-й, 13-й и 14-й недели разработки имели место потери персонала, за которым последовало резкое увеличение его количества. Будучи руководителем, следует вскрыть причину столь странного поведения показателя и выяснить, было ли это естественным отражением особенностей процесса разработки или же признаком наличия проблемы, которую необходимо решить немедленно, а не когда делать это будет уже поздно.

На рис. 11.3 показана диаграмма загрузки персонала по месяцам, на которой представлены следующие этапы: разработка требований (ТЗ), эскизное проектирование (ЭП), рабочее проектирование (РП), тестирование кода и модулей (ТКМ) и объединенные комплексные и системные испытания (КИ). Кроме того, в верхней части диаграммы помечены формальные пересмотры, проводимые в конце соответствующих этапов, а именно: заключение договора (ЗД), пересмотр эскизного проекта (ПЭП), критический пересмотр проекта (КПП), пересмотр готовности к тестированию (ПГТ) и приемочный пересмотр (Поставка). Месяцы, в течение которых разрабатываются требования, имеют отрицательную нумерацию, поскольку проводить какое-либо прогнозирование не возможно до тех пор, пока требования не определены, и, как правило, отсчет времени разработки начинается с момента заключения договора.

Обратите внимание, что на четвертом месяце диаграммы происходит переход от этапа ЭП к этапу РП, который длится около половины месяца. Именно в течение этого периода по плану должно быть произведен пересмотр эскизного проекта (ПЭП). Если учесть, что эта диаграмма была создана по окончании завершении проекта, то из сроков промежуточных этапов (включая поставку) становится ясно, что разработка этого проекта выполнялась в соответствии с календарным планом (промежуточные этапы приходятся на те же месяцы, в которых происходил переход от одного этапа к другому). Соответствие плану можно определить, наложив диаграмму с измененными сроками промежуточных этапов, и, следовательно, с измененными плановыми сроками, на диаграмму загрузки персонала.

На практике можно руководствоваться следующим эмпирическим правилом: смещение или удлинение плановых сроков можно считать номинальным, если это изменение не превышает 10%. Для плана проекта, предусматривающего выполнение проекта за 20 месяцев, завершение проекта в период между 18 и 22 месяцем оказалось бы номинальным. Изменения плановых сроков отдельных промежуточных этапов должны приводиться к ближайшему полумесячному сроку. Аналогично, если бы план предусматривал выполнение проекта в течение 20 недель, завершение разработки в период между 18 и 22 неделями было бы номинальным, причем изменения отдельных промежуточных этапов должны приводиться к ближайшему полунедельному сроку.

На рис. 11.4 приведено графическое представление измеренных данных показателя качества, а именно — количества обнаруженных ошибок на тысячу предполагаемых исходных строк кода (K of estimated source lines of code, KESLOC). Количество ошибок на тысячу строк кода, обнаруженное на каждом этапе жизненного цикла разработки, сообщается по завершении каждого из этих этапов. В данном случае этапы соответствуют этапам каскадной модели жизненного цикла разработки программного обеспечения, а именно разработке требований (ТЗ), эскизному проектированию (ЭП), рабочему проектированию (РП), тестированию кода и модулей (ТКАЛ), комплексным испытаниям (КИ), системным испытаниям (СИ) и приемочным испытаниям (ПСИ).

В случае больших программ количество ошибок может определяться отдельно для каждой категории серьезности ошибок и отображаться в виде накладывающих одна на другую столбчатых диаграмм. Анализ значений данных этой столбчатой диаграммы показывает, что данный проект является проектом, в котором применяется быстрое тестирование. Почему? 41% ошибок были обнаружены в ходе статического тестирования еще до начала этапа ТКМ, 36,3% ошибок были обнаружены на этапе ТКМ, и 22,7% — методами динамического и статического тестирования после завершения этапа ТКМ. Большинс-

тво ошибок (77%) было обнаружено и устранено на наиболее эффективном с точки зрения затрат этапе жизненного цикла разработки, а не после завершения этапа ТКМ.

На основе анализа этого конкретного случая SEPG и руководство института пришли к заключению о целесообразности того, что руководители всех проектов должны сообщать одни и те же четыре программных показателя (размер программы, количество занятого персонала, календарный план и качество). При этом они должны были делать это в графической форме, используя одни и те же определения и единицы измерений. Это позволило главному менеджеру и директорам сравнивать проекты между собой и задавать менеджерам разработки программного обеспечения более уместные вопросы.

На рис. 11.5 представлены двухуровневый набор показателей и программа составления отчетов, которые использовались в названной организации более 5 лет. Элементы верхнего уровня уже были полностью описаны. Соответствующие отчеты могут составляться на основных этапах, например, во время пересмотра эскизного проекта (ПЭП), критического пересмотра проекта (КПП), пересмотра готовности к тестированию (ПГТ) и по завершении заключительного тестирования качества (ЗТК). Нижний уровень представляет собственные показатели проекта, которые были отобраны, проанализированы и включены в отчет с целью удовлетворения специальных требований проекта.

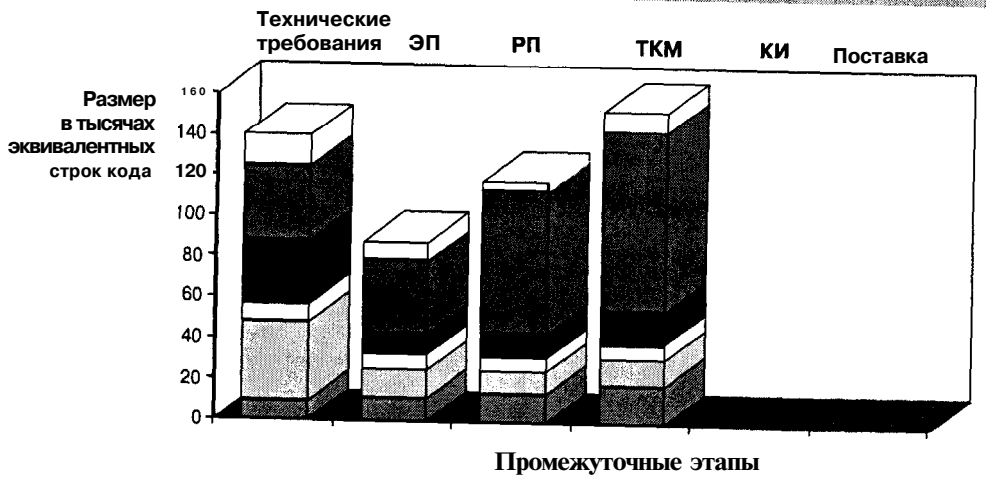



*Рис. 11.1. Диаграмма размеров и основные промежуточных этапов.*

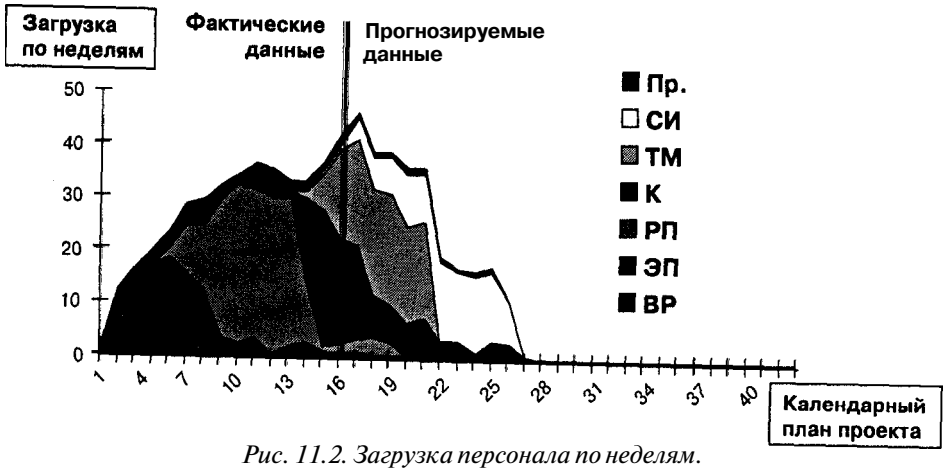



*Рис. 11.2. Загрузка персонала по неделям.*

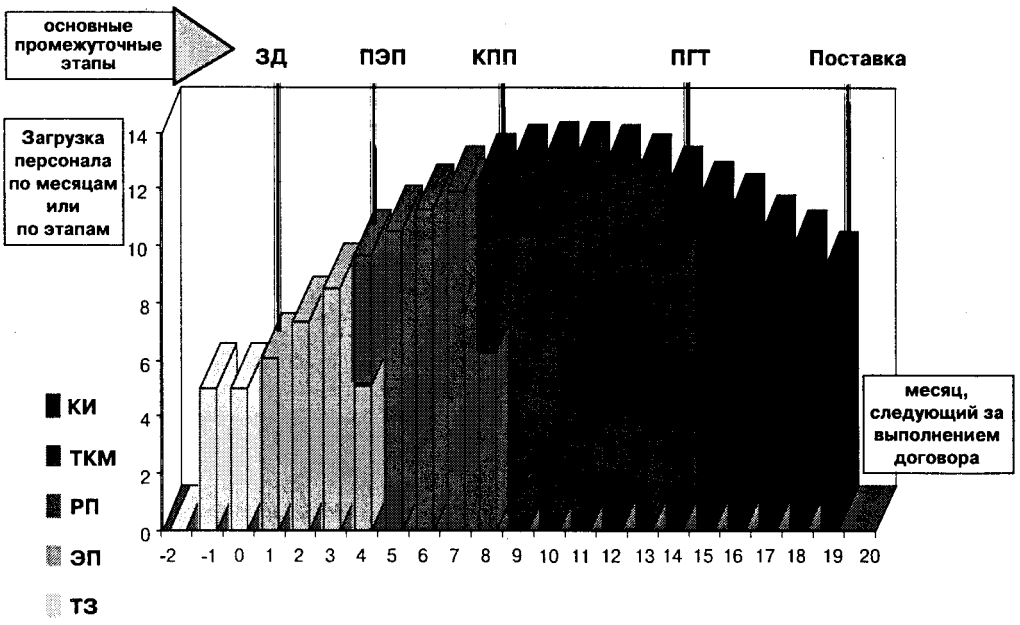



*Рис. 11.3. Сравнение фактических и плановых показателей.*

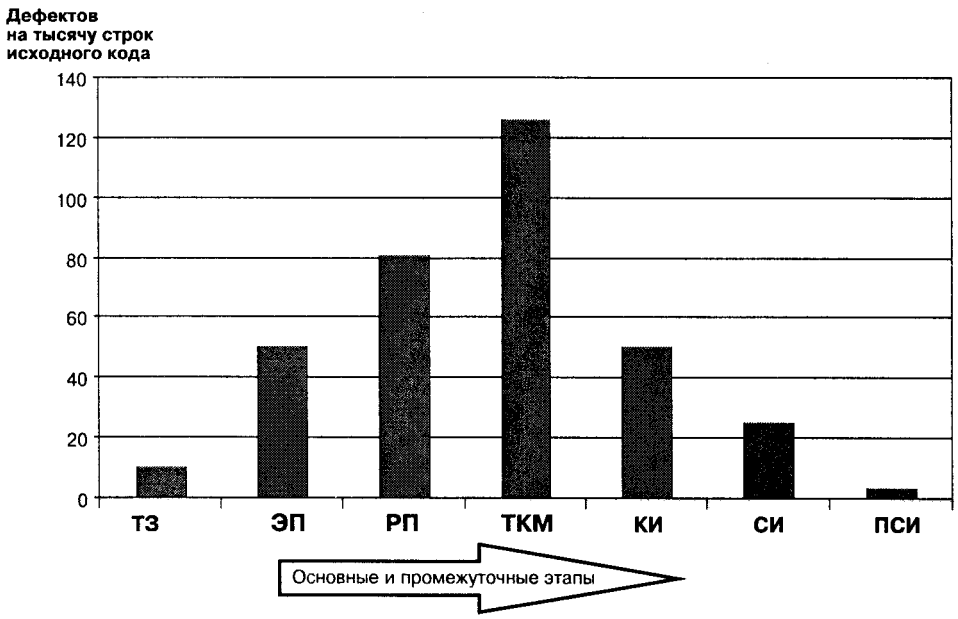



*Рис. 11.4. Распределение показателей качества по промежуточным этапам.*


*Рис. **11.5.** Двухуровневый набор показателей и программа составления отчетов.*

**В** следующем практическом примере мы познакомимся с технологией определения стандартного процесса, общего для всех проектов разработки программного обеспечения, которые выполняются в данной организации. Цель состоит не в том, чтобы во всех проектах использовались одни и те же единицы измерения конкретных показателей, а в том, чтобы предоставить персоналу, занятому разработкой, стандартный способ определения программных показателей, методик выполнения измерений и форм отчетности в рамках проекта.

### ПРИМЕР: ПРИМЕНЕНИЕ ПРОЦЕССА ЦВП ДЛЯ ОПРЕДЕЛЕНИЯ ПОКАЗАТЕЛЕЙ ПРОЕКТА (ГЭРИ КОББ)

В SEPG команды разработки были обучены применять специализированные показатели, соответствующие требованиям каждого отдельного проекта и клиента. Для членов команд был прочитан обучающий курс по процессу "цель-вопрос-показатель" (ЦВП), в основу которого были положены исследования Базили (Basili) и Вейса (Weiss) [6]. Метод ЦВП состоит в определении *целей (goal)* измерения каждого из показателей, формулировании *вопросов (questions),* на которые измерения должны дать ответ, и определении нормализованной формулы *показателя (metric),* который может масштабироваться в зависимости от размерности разработки. Примерами вопросов, относящихся к тестированию, ответы на которые могут быть получены в ходе выполнения процесса ЦВП, являются:

- Каково качество программного проекта?
- Каков фактический коэффициент переделок?
- Каков коэффициент закрытия отчетов о дефектах?

- Каково соотношение фактического коэффициента ошибок и прогнозируемого, нормативного и среднего значений этого показателя?
- Для каких компонентов потребуется дополнительное тестирование?
- С какими областями связано преобладающее большинство проблем?

Профессор Виктор Базили (Victor Basil!) и доктор Девид Вейс (David Weiss) из университета Мериленда сформулировали принцип ЦВП в конце семидесятых годов, работая над проектами в центре космических полетов Годдарда, NASA. Свою работу, посвященную ЦВП [6], они опубликовали в 1984 г. Рини ван Солинген (Rini van Solingen) и Эгон Бергхот (Egon Berghout) [50] продолжили разработку концепций ЦВП, сделав их составной частью принципа повышения качества (см. рис. 11.6).

На первом шаге команде разработки рекомендовалось записать цели, которые ставятся перед командой руководством института, организацией, осуществляющей руководство проектированием, лицами, принимающими решения, и любыми другими организациями (подразделениями), участвующими в управлении проектом. К характеристикам, которые могут быть сформулированы в качестве целей, могут относиться рентабельность, эффективность, качество продукта, степень удовлетворения требований клиента, объем программного кода и производительность разработки.

На втором шаге организуется совещание, в ходе которого занятый в разработке проекта персонал проводит "мозговой штурм" по перечню вопросов или проблем, которые должны быть выяснены, чтобы принимающие решения лица могли эффективно спланировать проект. Примерами вопросов, которые связаны с возможностью срыва плановых сроков, могут служить:

- Влияют ли последние изменения в бюджете на сроки выполнения тестирования?
- Соблюдаются ли плановые сроки тестирования?
- Какие события оказывают решающее влияние на действующий в настоящее время календарный график тестирования?
- Каковы последствия срыва календарного графика?

На третьем шаге предполагается обсуждение данных проекта, которые должны быть собраны (или сбор которых планируется), и того, можно ли на основе этих данных и их анализа получить ответы на вопросы, поставленные на втором шаге. Эти данные измерений могут собираться на уровне проекта или же быть внешними данными, такими как ресурсы, фактические затраты, сроки каждого из промежуточных этапов, распределение дефектов продукта по уровням серьезности или перечень сгруппированных по сборкам задач, которые должны быть решены. Таблица 11.1 содержит множество проектных показателей, относящихся к тестированию.

Подводя итоги анализа двух приведенных примеров, можно отметить следующее. Двухуровневая программа определения показателей была полностью одобрена всеми командами выполнения отдельных проектов. Им импонировало иметь свободу при создании собственных наборов показателей, по которым можно было составлять внутренние, отчеты или отказываться от их использования. Разделение функций между коллекцией показателей SEPG (для сбора данных измерений на общеинститутском уровне) и проектной коллекцией показателей (для сбора данных измерений на уровне отдельного проекта) оказалось ключевым фактором успешной деятельности института, его проектов и сохранения базы клиентов.

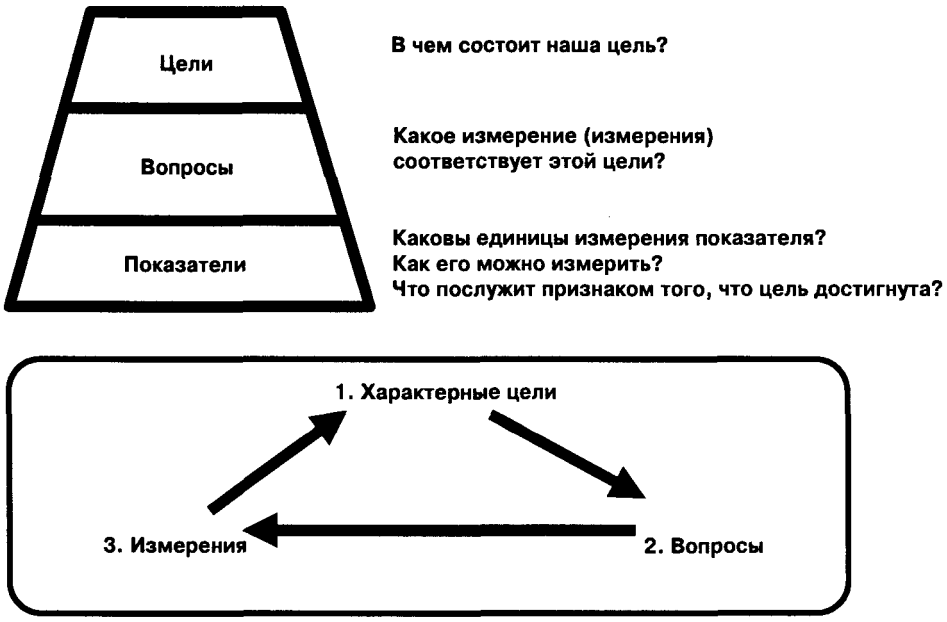



*Рис. 11.6. Принцип "целъ-вопрос-показатпелъ"* ***(ЦВП).***

# Использование стандартных показателей для внесения усовершенствований

Простое выполнение в рамках одной организации множества проектов по разработке, направленных на совершенствование процесса, еще не гарантирует создание профаммного обеспечения мирового класса. Если каждая команда выбирает собственное направление совершенствования процесса или продукта, оптимизация процесса будет носить случайный характер. В этом случае локальная оптимизация выливается в одно из направлений глобальной оптимизации. Для обеспечения быстрой оптимизации профаммного обеспечения усилия должны быть согласованы в рамках всей организации с соблюдением общего для нее подхода. Успешность совершенствования управления процессом разработки программного обеспечения определяется следующими факторами:

- Точным прогнозированием взаимосвязи между функциональными возможностями программного обеспечения и его объемом, количеством занятого в разработке проекта персонала, календарным графиком и качеством проекта.
- Определением задач и ресурсов/качества/стандартов, необходимых для решения каждой задачи.
- Обеспечением успешности и требуемой производительности при выполнении каждого шага процесса.
- Максимально быстрым выявлением дефектов после их появления.
- Обработкой дефектов таким образом, чтобы обеспечить максимально быстрое эффективное устранение наиболее важных дефектов.

Рекомендуемая стратегия создания сопровождающей документации по этой методике определения программных показателей состоит из двух частей:

1. Применение стандартного набора определений показателей и последующий сбор и анализ данных программных измерений в рамках всех выполняемых в организации проектов.
2. Применение процесса принятия решений "цель-вопрос-показатель" для выбора проектных программных измерений.

Одно из наиболее эффективных действий, ведущих к совершенствованию процесса — это управление способом прогнозирования ресурсов с целью повышения его точности. Если прогнозируемые значения нанести на диаграмму процесса разработки с привязкой к каждому из основных промежуточных этапов, после чего сравнить совпадение кривых обучения с фактическими данными, можно отслеживать точность прогнозирования. Пример применения этой технологии к прогнозированию загрузки персонала показан на рис. 11.7, который взят из [6]. Прогнозируемые данные приводятся к среднему значению A и наносятся на диаграмму, по горизонтальной оси которой отсчитывается время в месяцах. Круглые точки представляют нормализованные прогнозируемые значения, причем последнее прогнозируемое значение приходится на конец 7-го месяца. На диаграмму можно наложить кривые функций U и L (обозначенные квадратиками) и тем самым определить показатель экспоненты B. Чем выше значение B, тем быстрее прогнозируемые значения начинают совпадать с фактическими.

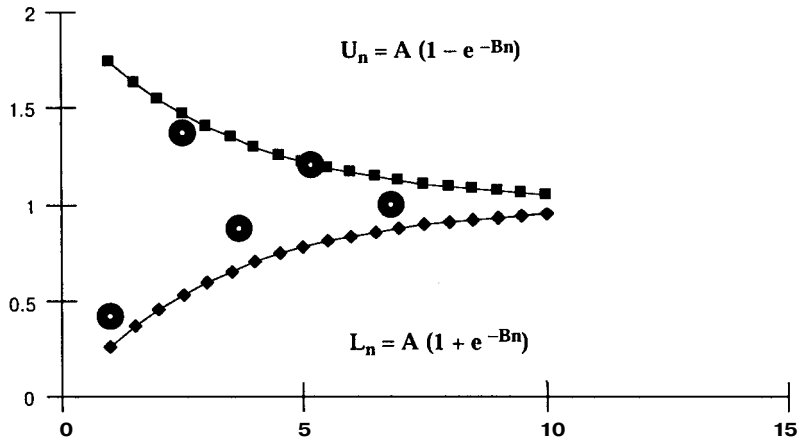



***Рис. 11.7. Совпадение кривых обучения с нормализованными прогнозируемыми данными.***

На основе значений, полученных в ходе предыдущих разработок, можно создать обеспечивающую приемлемую точность модель прогнозирования количества персонала/календарных сроков разработки программного обеспечения, которая представляется нелинейной функцией от объема программы. Результаты корректно выполненного моделирования могут избавить руководителей от сложных проблем, возникающих при использовании простой линейной модели прогнозирования трудоемкости и времени, требующегося для выполнения проекта. Например, в соответствии с простой линейной моделью, если новая разработка вдвое превосходит по объему предыдущую, ее бюджет должен вдвое превосходить стоимость предыдущей разработки. В действительности же зависимость между затратами и объемом разработки не линейна, и при использовании неподходящей модели ошибка прогнозирования может оказаться весьма значительной.

Прежде чем можно будет воспользоваться нелинейной моделью зависимости от объема программного обеспечения, потребуется выполнить следующие четыре задачи:

1. Собрать данные, полученные в ходе выполнения предшествующих проектов, аналогичных оцениваемому. Сюда должны входить и данные, относящиеся к объему проекта, фактическим финансовым показателям, календарным срокам и факторам среды разработки.

2. По аналогии с предыдущими проектами разработать оценки, сравнивая трудоемкость, фактические финансовые показатели, данные программных показателей и факторы среды разработки нового проекта с соответствующими показателями аналогичных проектов.

3. Создать календарный график выполнения основных промежуточных этапов, на базе которого создать календарный график выполнения всего проекта по месяцам.

4. Завершить оценку, определив количество персонала, необходимого для выполнения каждой задачи, и распределить загрузку персонала по соответствующим этапам календарного графика разработки.

Сообщение всеми группами разработки таких данных измерений, как показатели объема, занятости персонала, стоимости и качества, в центральную организацию наподобие SEPG обеспечивает проверку различных проектов на предмет единообразия выполнения. Кроме того, эти данные измерений поддерживают нелинейное моделирование общеорганизационных процессов разработки, которые определены на основе фактических данных измерений, полученных в ходе выполнения предыдущих проектов в аналогичной среде разработки. Точное прогнозирование характеристик новых проектов направлено на резкое снижение хаоса в организации, и сбор и анализ данных измерений этих стандартных показателей в значительной степени способствует достижению такой цели. Общий для конкретной организации процесс управления данными измерений должен включать в себя каждый из пяти видов деятельности, перечисленных в таблице 11.3.

**Таблица 11.3. Виды деятельности по подготовке к прогнозированию и их определения**

| Виды деятельности | Определения |
|---|---|
| Сбор/упорядочение данных | Прямые данные измерений собираются, а производные данные измерений вычисляются, после чего они форматируются для внесения в базу данных. |
| Сохранение/ извлечение данных | Прямые и производные данные измерений сохраняются в базе данных ранее выполненных разработок, которая служит основным источником исходной информации при принятии решений. |
| Анализ данных | Данные измерений исследуются для выяснения тенденций и отклонений от среднестатистических данных (значительных отклонений от ожидаемых норм). |
| Обобщение информации | Данные измерений и анализа моделируются, |
| Распространение информации | Обобщенные отчеты (о состоянии, прогнозы, анализы вероятности возникновения проблем) распространяются среди лиц, принимающих решения. |

Если проекты разработки программного обеспечения поддерживаются общеорганизационными функциями сбора и анализа данных измерений, это позволяет управлять совершенствованием процесса и сравнивать его результаты с результатами, полученными в ходе ранее выполнявшихся разработок. В результате снижается доля специализированных или скоропалительных изменений, которые препятствуют организации повышать качество или сокращать сроки поставки программных продуктов на рынок. Календарные сроки, качество и затраты — основные показатели коммерческой деятельности. Поэтому, чтобы организация, занимающаяся разработкой программного обеспечения, смогла выжить в условиях конкуренции, процессы создания программного обеспечения должны оцениваться именно с таких коммерческих позиций. Принцип быстрого тестирования обеспечивает совершенствование процесса в следующих областях:

- Календарного планирования — путем выделения дополнительных ресурсов для действий по тестированию до начала этапа TKM, что позволяет ужать календарные сроки выполнения действий по тестированию на последующих этапах и способствует ускорению поставки на рынок очередных выпусков программных продуктов.
- Качества — путем обнаружения и устранения большего количества дефектов, что снижает процент скрытых дефектов в выпущенных продуктах.
- Стоимости — путем снижения одноразовых затрат, объем которых выше в тех проектах, где обнаружение и устранение дефектов откладывается до заключительных этапов жизненного цикла.

# Показатели тестирования

Если вспомнить концепцию двухуровневой программы определения показателей, может возникнуть вопрос о ее применимости к показателям тестирования. Относятся ли показатели тестирования к более высокому, общеорганизационному уровню, или же к проектно-зависимому уровню этой двухуровневой программы? Как будет показано в этом разделе, почти все показатели тестирования относятся к категории проектно-зависимых показателей. Примеры показателей тестирования, используемые в проектах, приведены в таблице 11.4.

**Таблица 11.4. Категории показателей тестирования с примерами**

| Показатель тестирования | Определение |
|---|---|
| Ошибки предикативной логики | Количество ошибок, обнаруженных в утверждениях логических конструкций (IF, WHILE, CASE и т.п.), приходящихся на тысячу эквивалентных строк кода. |
| Ошибки индексирования/ пределов циклов | Количество ошибок, обнаруженных в индексировании или пределах циклических конструкций (DO, FOR и т.п.) приходящихся на тысячу эквивалентных строк кода. |
| Прогресс тестирования | Сравнение количества выполненных случаев тестирования и их общего запланированного количества на определенном промежутке времени (характеризует способность поддерживать заданный прогресс тестирования). |
| Заявки на изменения и изменчивость элементарных действий | Количество оговоренных изменений продукта и направлений разработки (может служить признаком влияния на календарный план и затраты). |
| Отсутствующие/неполные требования | Количество обнаруженных дефектов, появившихся в результате отсутствующих или неполных требований. |
| Прогресс проектирования | Количество задокументированных требований — запланированное по сравнению с фактическим (служит мерой способности поддерживать прогресс проектирования на этом раннем этапе). |

Разработка эффективной программы определения показателей требует строгой дисциплины при выполнении измерений, анализе результатов этих измерений и построении отчета о тенденциях, о которых свидетельствует анализ. Требуемые для этого усилия и дисциплина могут существенно увеличить трудоемкость программы разработки показателей, но эти дополнительные усилия очень важны для успеха любой инициативы по совершенствованию процесса. Когда изменения для совершенствования процесса вносятся в ходе выполнения проекта, измерения будут единственным способом определить ценность проделанных усовершенствований процесса. При сравнении одного проекта с другим сравнение полученных в ходе выполнения проекта данных — единственный способ выявления различий между проектами.

## Плотность ошибок (количество ошибок на тысячу эквивалентных строк кода)

Наиболее важные показатели тестирования связаны с ошибками (дефектами). Для проекта представляется вполне естественным выполнение конкретизации ошибок, разделение их по приоритетам и категориям. Как правило, разделение ошибок по приоритетам выполняется в соответствии с нормативами по определению уровня серьезности ошибок, как показано в таблице 11.5. Обратите внимание, что номер уровня серьезности присваивается таким образом, что 1 соответствует категории катастрофических ошибок, а 5 — категории неприятных ошибок. Определения серьезности, перечисленные в таблице 11.5, совпадают с определениями из таблицы 5.2 в главе 5.

**Таблица 11.5. Код серьезности**

| Номер уровня серьезности | Нормативы |
|---|---|
| 1 | Катастрофический — приводит к отказу системы, т.е. к пустому экрану или повреждению данных. |
| 2 | Серьезный — продукт не пригоден к использованию, т.е. приводит к ошибочным ответам, неправильным отчетам. |
| 3 | Умеренный — продукт пригоден к использованию, но дефект оказывает влияние на действия, выполняемые клиентом. |
| 4 | Незначительный — продукт пригоден к использованию, дефект не оказывает влияния на действия, выполняемые клиентом. |
| 5 | Помеха — дефект может быть устранен в любое удобное время. |

Сбор данных о дефектах и создание гистограммы с использованием номеров серьезности, присвоенных каждому дефекту, позволяет аналитику описать качество выпуска программы, который будет проверяться во время формальных пересмотров. Пример гистограммы плотности ошибок, сгруппированных по номерам уровня серьезности, показан на рис. 11.8. Обратите внимание, что эти данные приведены к ожидаемому объему программы, выраженному в тысячах строк эквивалентного кода.

Из приведенных на рис. 11.8 данных видно, что большой объем ошибок был обнаружен на ранних этапах жизненного цикла разработки. Из этих данных понятно, **что** для обнаружения ошибок на этапах разработки требований, проектирования и кодирования применялось статическое тестирование. Обратите также внимание, что на этапах РП и кодирования было обнаружено больше серьезных ошибок, чем на этапе КИ. Это означает, что тщательное тестирование продукта было начато задолго до этапа КИ — подход, который является отличительной чертой быстрого тестирования. Если создать программу определения показателей, которая обеспечивает подсчет и составление отчетов об ошибках, обнаруженных на каждом из этапов жизненного цикла разработки, можно оценить и отобразить эффективность усовершенствований процесса по мере их внесения от проекта к проекту.

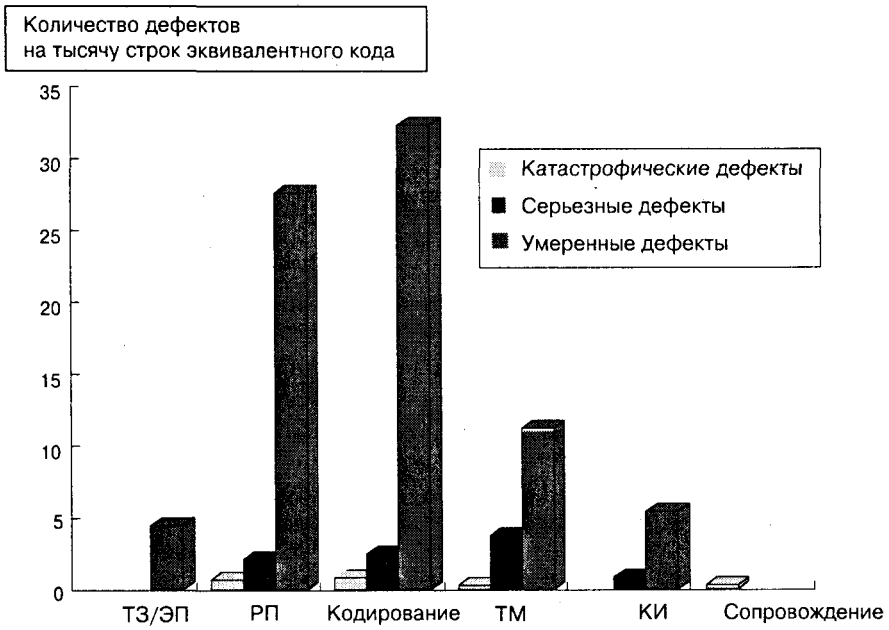



*Рис. 11.8. Гистограмма плотности ошибок, сгруппированных по нескольким номерам уровня серьезности.*

# Проектно-ориентированная модель ошибок

Отдельные языки программирования могут быть более чувствительны к некоторым типам ошибок, чем другие. Например, какой-либо язык программирования может генерировать код, который особенно трудно сопровождать или повторно использовать в других программах. Код, генерируемый другим языком, может оказаться более пригодным для Сопровождения или повторного использования, но может характеризоваться более низкой производительностью. Один из способов определения чувствительности проекта к различным типам ошибок предполагает создание модели ошибок для данного проекта. Проекты, в которых используется один и тот же язык программирования, могут сравниваться, только если в них используется та же самая модель ошибок. Если при сравнении с моделью ошибок выясняется, что в данном проекте вероятность возникновения конкретного типа ошибок высока, это может служить признаком наличия систематической проблемы в методе разработки программного обеспечения. Иначе говоря, анализ данных ошибок скорее ведет к обнаружению просчетов в процессе разработки, а не в самом продукте.

Перечень известных источников ошибок приведен в таблице 11.6. Ошибки могут быть разделены по категориям в ходе процесса пересмотров, когда комиссия приходит к согласию относительно категории основной причины проблемы. Например, если основная причина относится к категории ошибок прослеживаемости (TR), комиссия по пересмотру определяет этап жизненного цикла проекта, на котором эта ошибка была внесена (например, этап эскизного проектирования (ЭП)). Эти категории могут быть представлены графически и проанализированы, как показано на

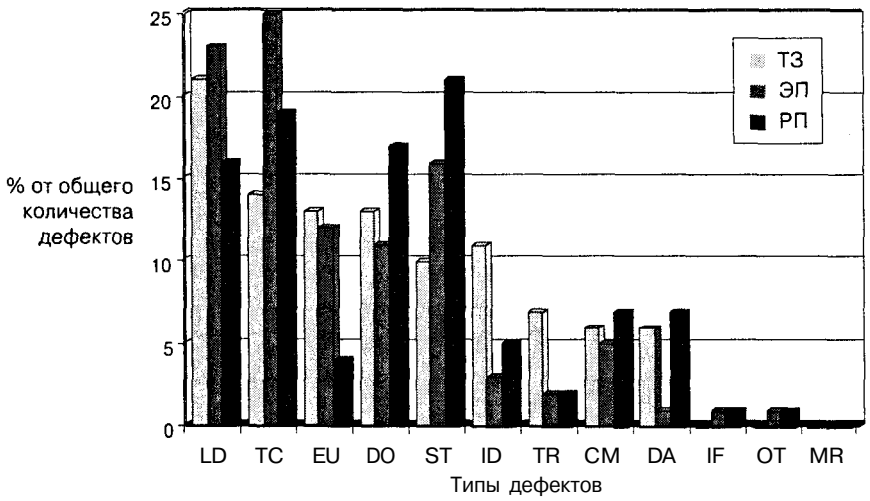
рис.11.9. Гистограмма, полученная в результате анализа, будет характерной для данного языка программирования и для данного жизненного цикла программного обеспечения.

**Таблица 11.6. Категории ошибок проекта**

| Код | Название категории ошибки | Определение категории |
|---|---|---|
| **LD** | Уровень детализации | Отклонение от надлежащего уровня детализации: слишком подробно, недостаточно подробно, неполно. |
| **TC** | Техническое содержание | Техническое содержание неверно, допускает двоякое толкование, не соответствует стандартам. |
| **EU** | Применение языка | Ошибки орфографии, грамматики, пунктуации, употребления времен и другие ошибки применения языка. |
| **DO** | Документация | Документация не соответствует выполнению программы или другой документации. |
| **ST** | Стандарты | Несоответствие установленным стандартам форматов или процедур. К этой категории не относятся ошибки несоответствия стандартам синтаксиса применения языка, орфографии или читабельности. |
| ID | Унаследованные | Ошибки в повторно используемом или существующем коде. |
| TR | Прослеживаемость | Несоответствие ранее выпущенным документам; противоречия, недостаточное количество ссылок, неправильные ссылки. |
| CM | Полнота | Конечный продукт или его документация требует дополнительной информации или реализации дополнительных функциональных возможностей. |
| DA | Данные | Отсутствие или ввод неправильных данных пользователем или из базы данных. |
| IF | Интерфейс | Ошибки в интерфейсе. |
| OT | Прочие | Ошибки, которые не могут быть отнесены ни к одной из вышеуказанных категорий. |
| MR | Пригодность для сопровождения и повторного использования | Сложность сопровождения или повторного использования; не достигнут компромисс между высокой сложностью и гибкостью. |

Еще одно преимущество анализа данных ошибок состоит в возможности автоматического сравнение следующего проекта с предыдущим. Это ведет к двум важным последствиям: во-первых, в ходе пересмотров внимание можно сосредоточить на обнаружении наиболее вероятных ошибок и, во-вторых, можно иметь представление о том, оказывает ли некоторое изменение в процессе желаемое влияние на снижение вероятности определенного типа ошибок.

Вообще говоря, анализ основных причин возникновения ошибок, подобный выполненному при определении модели ошибок проекта, является мощным средством совершенствования процесса. Идентификация проблем в процессе разработки, их устранение и оценка эффективности процесса устранения представляют собой основные цели создания программы для выбора жестких показателей тестирования.




| | | | | | |
|---|---|---|---|---|---|
| LD | Уровень детализации | ST | Стандарты | DA | Данные |
| TC | Техническое содержание | ID | Унаследованные | IF | Интерфейс |
| EU | Применение языка | TR | Прослеживаемость | OT | Прочие |
| DO | Документация | CM | Полнота | MR | Пригодность для сопровождения и повторного использования |

*Рис. **11.9.** Проектно-ориентированная модель ошибок.*

# Программа оценки ошибок программного обеспечения (SWEEP)

Программа оценки ошибочности программного обеспечения (Software Error Estimation Program, SWEEP), разработанная консорциумом Software Productivity Consortium, является средством прогнозирования для расчета коэффициентов ошибок, в том числе и скрытых. Скрытой ошибкой считается любая ошибка, остающаяся в программном продукте, т.е. такая ошибка, которая существует, но еще не обнаружена. Ценность модели SWEEP состоит в том, что она упрощает управление и прогнозирование ошибок в системах с интенсивным использованием программного обеспечения. Эта модель поддерживает определение целей обнаружения ошибок во время разработки программного обеспечения и помогает отслеживать продвижение по пути реализации этих целей. За счет прогнозирования количества ошибок, остающихся в программной системе, SWEEP осуществляет мониторинг и помогает контролировать качество программных продуктов. Ниже приведен перечень допущений, лежащих в основе модели:

- Все обнаруженные ошибки должны быть записаны в момент их обнаружения.
- Ошибки исправляются после их обнаружения, и во время устранения обнаруженных ошибок не вносятся новые ошибки.
- Трассировка ошибок должна выполняться единообразно в течение всех этапов жизненного цикла.

- Ошибки в документации должны отслеживаться отдельно от программных ошибок.
- Входные данные программы SWEEP должны регулярно проверяться и обновляться.
- Точность результатов выполнения программы SWEEP будет повышаться пропорционально вводу фактических количеств ошибок, полученных на других этапах процесса разработки.

Определения трех режимов использования программы SWEEP приведены в таблице 11.7.

**Таблица 11.7. Определения режимов программы SWEEP**

| Режим | Название | Особенности |
|---|---|---|
| 1 | Модель с привязкой ко времени | Позволяет оценивать и отслеживать ошибки на этапах системных и комплексных испытаний. |
| 2 | Модель с привязкой к этапам | Позволяет прогнозировать и отслеживать ошибки на нескольких этапах и может предоставить информацию об ошибках до выполнения какого-либо кода. |
| 3 | Модель поддержки планирования | Дает возможность пользователю определить цели обнаружения ошибок в программном проекте, исходя из опыта предыдущих проектов. Затем на основе ранее собранных данных модель генерирует профиль обнаружения ошибок, помогающий в достижении сформулированных целей. |

Тестировщикам программного обеспечения SWEEP предлагает понятное графическое представление обнаруженных ошибок и прогноза оставшихся ошибок. Кроме того, она дает возможность пользователям выполнять сравнение с ранее полученными данными, что делает реальной точную оценку состава скрытых ошибок на различных этапах жизненного цикла. Руководителю подразделения обеспечения качества программа SWEEP позволяет точно прогнозировать количество циклов тестирования, которые потребуются для удовлетворения требований заданного стандарта качества. Пример модели SWEEP, в которой используется привязка к этапам, приведен на рис. 11.10. Обратите внимание, что в данном проекте только что завершилась инспекция начального исходного кода на предмет его соответствия рабочей проектной документации, в ходе которой было обнаружено 3,48 ошибок на тысячу строк кода. Наибольшее совпадение рэлеевскои кривой распределения ошибок с фактическими данными наблюдается для момента получения самых последних данных, и ему соответствует значение, равное 3,48. Значения 3,3 и 7,8, полученные, соответственно она этапах эскизного и рабочего проектирования, проходят через это же значение 3,48. В соответствии с этой рэлеевскои кривой распределения на следующем этапе тестирования блоков следует ожидать обнаружения около 2,12 ошибок на тысячу строк исходного кода. В случае сохранения той же тенденции в момент поставки клиенту программный продукт должен содержать менее 0,07 ошибок на тысячу строк кода.

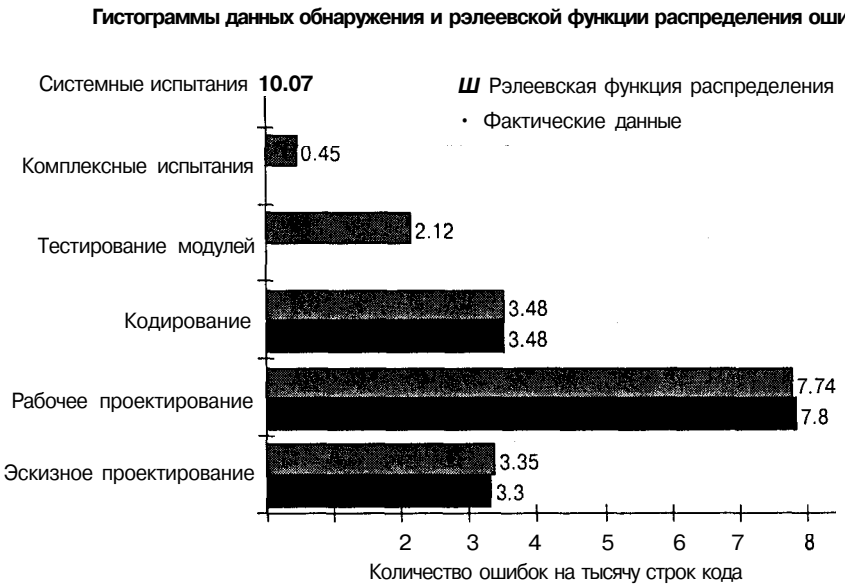



***Рис. 11.10. Режим 2 SWEEP:*** *модель с привязкой ко времени*

Следующим логическим шагом процесса было бы усреднение для большого числа проектов фактических данных о количестве ошибок, приходящихся на тысячу строк кода, которые обнаружены на каждом из этапов (ЭП, РП, ТКМ, КИ и СИ). В следующем проекте должна иметься возможность использования этих данных, обозначенных на рис. **11.11** как "Номинальные значения", в качестве меры количества ошибок, обнаружение которых следует ожидать при выполнении проекта с таким же уровнем качества, или даже меры прекращения дальнейшего тестирования во имя повышения производительности инспекций. Верхний и нижний пределы уровня контроля качества могут быть установлены, например, равными трем сигма от номинальных значений данных.

Рэлеевская функция распределения ошибок представляет собой важный инструмент в арсенале инженера системного тестирования. На этапе системного тестирования все существующие случаи тестирования применяются методично день за днем, в ходе выполнения тестирования новых подсистем и регрессивного тестирования новых сборок, в которых устранены ошибки, обнаруженные в предыдущих сборках. На рис. 11.12 показаны фактические данные ошибок, собранные в ходе выполнения ряда системных испытаний кандидатов на роль выпусков. При этом сборки выполнялись еженедельно, а обнаруженные ошибки наносились на столбчатую диаграмму ежедневно. Вычерчивание огибающей было выполнено автоматически при помощи модели с использованием привязки ко времени программы SWEEP. Обратите внимание, что если заранее было определено допустимое количество ошибок на тысячу строк кода, которое может быть поставлено клиенту, тестировщики могут смело использовать эту диаграмму для определения условий прекращения этапа системных испытаний.

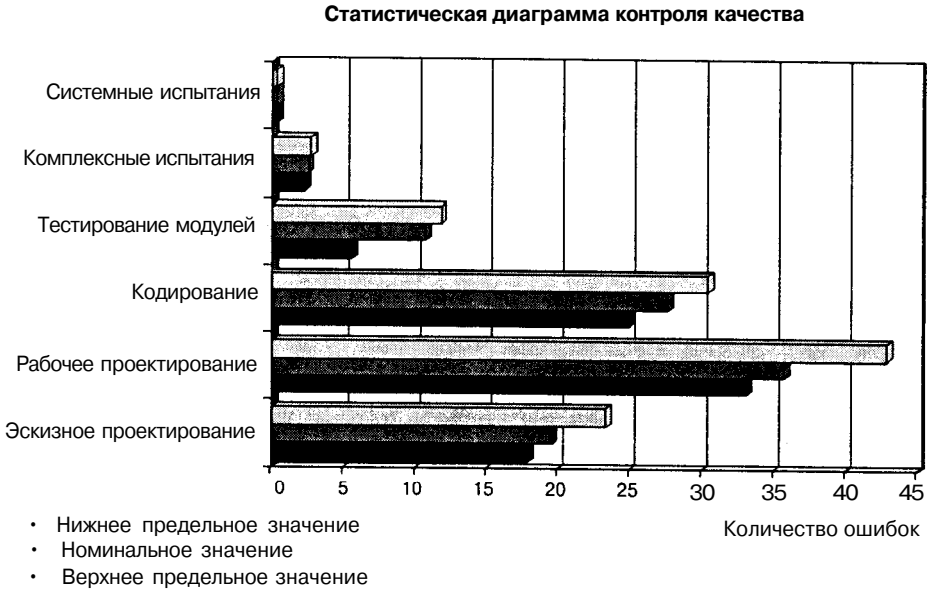



*Рис. 11.11. Совмещенная модель поддержки планирования и модель с привязкой к этапам программы SWEEP.*

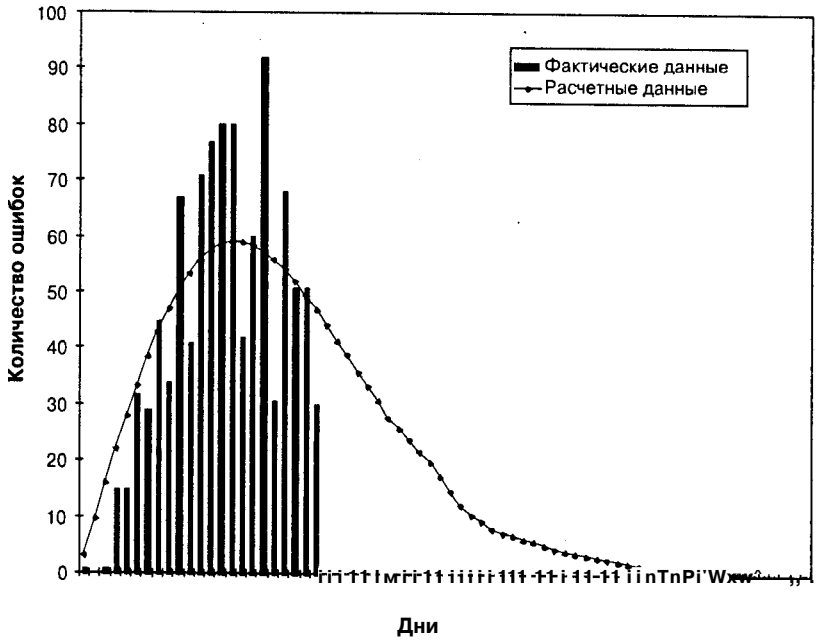



*Рис. 11.12. Использование модели с привязкой ко времени программы SWEEP для оценки данных по ошибкам на этапе системных испытаний.*

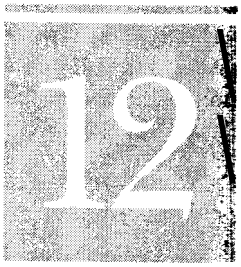
# Резюме

Проблемы и их решения, выбранные для исследования в этой главе, связаны с применением программных показателей к выполнению тестирования как с учетом интересов данного проекта, так и с учетом интереса всей организации в целом. Ниже приведен краткий перечень вопросов, рассмотренных в главе:

- Различия между показателями и данными измерений.
- Двухуровневый подход к сбору и анализу данных измерений.
- Полезные общеорганизационные показатели, а именно: показатели объема, трудоемкости, календарного графика и качества.
- Принцип выбора показателей уровня проекта, названный принципом "цель-вопрос-показатель" (ЦВП).
- Использование показателей для управления и совершенствования процесса разработки программного обеспечения.
- Перечень показателей тестирования (относящихся ко второму уровню).
- Разработка модели ошибок, которая может применяться при отслеживании процесса и повышении качества.
- Определение конечной цели тестирования, а именно: применение прогнозирования плотности ошибок, использование программы SWEEP и достижение конечного качества, т.е. количества скрытых ошибок, приходящегося на тысячу строк кода, которые поставляются конечному пользователю.

Во многих успешных проектах показатели используются для повышения наглядности и контролепригодности календарного графика, характеристик качества и стоимости. Лишь в отдельных неудачных проектах применение показателей приводило к разрушению доверительных отношений, столь необходимых во время разработки программного обеспечения. Основной причиной возникновения сбоев в программах определения показателей проекта практически всегда является "поиск виновных". Поиск виновных в ошибке (будь то отдельный исполнитель или группа лиц, принимавших участие в разработке) вовсе не является целью применения показателей. Они предназначены исключительно для эффективного управления разработкой программного обеспечения и качеством продукта.

# Технологии оценки трудозатрат на тестирование и советы




Темы, рассматриваемые в главе:

- Применение математических методов для оценки программного обеспечения
- Технология функциональных баллов
- Резюме

Целью настоящей главы является описание точного метода прогнозирования кадрового обеспечения по месяцам для подготовки тестирования и прогона тестов, которое обычно называется верификацией и аттестацией (verification and validation. V&V). Выбранный нами подход основан на изучении последовательности примеров, в рамках которых используется набор инструментальных средств, разработанных авторами.

Существует целый ряд методов построения графиков работ и календарного планирования ресурсов для проведения тестирования проекта. Простейший метод получил название прогнозирование с использованием модели с базисом оценок (basis of estimates, BOEs). Оценка по аналогии (by-analogy estimate) основана на фактических данных о распределении ресурсов на некотором множестве прошлых проектов аналогичной архитектуры, с одними и теми же языками программирования и аналогичными подходами к тестированию. Оценки будущего проекта, если в их основу положены оценки по аналогии, строятся на правдоподобии за счет использования эмпирических фактов с инкрементальным совершенствованием, которые основаны на объединении решений мелкого масштаба. Инкрементальное совершенствование увеличивает точность прогнозирования. В число предлагаемых примеров входит классификация программных продуктов по их размерам и оценка ресурсов групп поддержки, например, группы обеспечения качества, группы управления конфигурацией и руководство разработкой программы.

### ПРИМЕР 1. ПОДХОД К ОЦЕНКЕ (ГЭРИ КОББ)

Одним прекрасным утром я получил по электронной почте письмо, которое начиналось так: "Можете ли вы предложить какие-либо методы оценки объемов трудозатрат и необходимых ресурсов для проведения работ по тестированию программного обеспечения, а также для расчета графика этих работ?" Далее в письме говорилось: "Наша группа разработала таблицу результатов, в которую сводятся результаты измерений характеристик нашего процесса и прогнозы относительно текущего состояния проекта в целом, в зависимости от которых ему присваивается "красное", "желтое" или "зеленое" состояние. Чтобы осуществить это мероприятие, мы назначили некоторым из наших групп статус ведущих, в то время как другие получили такие названия, как, например, консультант PCQA (Product Certification and Quality Assessment — сертификация и управление качеством программного продукта). Принимая во внимание состояние наших последних разработок, можете ли вы предложить какой-то способ оценки объемов трудозатрат и ресурсов, необходимых для проведения работ по тестированию программного обеспечения, а также для расчета графика этих работ?"

В ответ я направил следующее предложение: метод оценки качества программного обеспечения, который, по моему мнению, можно будет внедрить в вашем подразделении, принимает следующий вид (за цель принимается уровень качества):

1. Разбейте весь персонал разработчиков по исполняемым каждым сотрудником ролям (т.е. специалисты по формулированию технических требований, архитекторы программного обеспечения, разработчики, кодировщики, тестировщики, руководитель разработки, персонал, управляющий конфигурацией средств тестирования, персонал, выполняющий сертификацию программного продукта, группа контроля качества и другие).
2. Воспроизведите полный рабочий график двух последних успешно завершенных проектов, например, поставленных заказчику в установленные сроки, без перерасхода сметной стоимости и приемлемого качества, и распределите временной ресурс и финансовую смету по ролям, определенным в пункте 1.
3. Вычислите величину показателя LOE (Level Of Effort — уровень трудозатрат) для каждой роли, т.е. LOE(j) для каждого j = 1,...,n, где n есть число ролей, выраженное через временной эквивалент FTE (Full-Time-Equivalent — эквивалент полного рабочего дня) в человеко-месяцах.
4. Вычислите сумму S в человеко-месяцах для каждого из этих показателей LOE по каждому проекту. Подсчитайте, какая доля *в* процентном отношении от общих трудозатрат приходится на каждую такую роль, например, как LOE(j)/S, для j = 1. . . . л в рамках каждого проекта, и сравните обе таблицы процентных отношений.
5. Вычислите среднее значение LOE(j)/S, для j = 1,...,n для обоих проектов, чтобы калибровать модель значений LOE(j), предназначенную для прогнозирования значений LOE(j) для новых проектов.
6. Оцените размер программного продукта в тысячах строк KESLOC (Estimated Source Lines Of Codes — предполагаемое количество строк исходного кода) для каждого проекта. KESLOC можно получить путем прямого подсчета строк исполняемого кода, отличных от комментариев, включая строки всех сценариев, которые не являются частью продукта, но были разработаны для вспомогательных целей. В качестве альтернативы можно подсчитать все функциональные баллы (рассмотренные далее в главе) и при помощи таблицы перевода Кейперса Джонса (Capers Jones), которая приводится в таблице 12.1, получить KESLOC для функциональных баллов.
7. Вычислите коэффициент EAF (Effort Adjustment Factor — коэффициент уточнения трудозатрат) из уравнения EAF = S/(2.4*(KESLOC**1,05)). Отсюда может быть выведена формула перевода размера продукта в трудозатраты:

   S = EAF*2.4*(KESLOC**1,05),

   где EAF получена на основании базиса оценок, а KESLOC прогнозируется на основе задокументированных требований.

**На мой первый ответ по электронной почте пришло письмо следующего содержания: "Я благодарю вас за то, что вы не пожалели своего времени, чтобы ответить мне, однако ваш ответ привел меня в еще большее замешательство. Вот почему я так долго не отвечал вам — я даже не знаю, что больше всего меня озадачило. Предложенный вами подход хорош, однако он предполагает, что необходимые данные находятся где-то рядом. В то же время у нас нет доступа к данным о трудозатратах, а то, что у нас есть, особой ценности не представляет. Единственное, что приходит в голову, так это найти аналогичный завершенный проект и воспользоваться фактическим количеством строк кода в нем как оценкой текущего проекта. Находите ли вы мое суждение имеющим смысл?"**

**Я отправил по электронной почте второе послание, в котором заметил: "Да, это один из способов стабилизации вашей оценки, например, с помощью базиса оценки программного кода предыдущего проекта. Вы также можете собрать меньшие оценки уровней LOE для используемого набора ролей и количество исполнителей на каждой роли, *на*пример, число руководителей проектов, программистов, тестировщиков различных специальностей, специалистов по управлению конфигурациями, лиц, ответственных за качество программного продукта, специалистов по подготовке сборок, по проведению испытаний, лиц, осуществляющих контроль производством и прочих. Сюда следует включить и расширенные группы, такие как группа подготовки документации, группа инструкторов, занимающихся обучением пользователей, группа специалистов, проводящих пробные испытания, группа экономистов, решающих вопросы конъюнктуры, группа субподрядчиков и т.п. Если вы располагаете такими сведениями, разделите соответствующие значения на общее число сотрудников, чтобы получить процентное отношение трудозатрат, а затем воспользуйтесь этими данным для нового подсчета трудозатрат. Точность этих расчетов находится в пределах 30%. Между прочим, 30% — это, на мой взгляд, несколько лучше, чем ошибки в 2 или даже в 4 раза, которые допускают разработчики программного обеспечения, раздувая сметную стоимость и трудозатраты. Удачи."**

Формулы, используемые в этом исследовании, заимствованы из модели COCOMO, предложенной Боемом (Boehm). Описанные выше семь шагов используют для получения коэффициента EAF скорее метод инженерного анализа, а не подход д-ра Боема, описание которого можно будет найти далее в главе.

При получении оценок возможны просчеты, причина которых кроется в статистических данных за прошлый период:

- В предыдущем и текущем проектах могли использоваться различные языки программирования, различные этапы жизненного цикла, различные технологии экспертных оценок и т.д. Из-за несоответствий в упомянутых областях применение статистических данных может привести к получению неточных рабочих графиков и оценок затрат.
- Часто трудно противиться желанию вычислять временные затраты и трудозатраты, исходя из предположения их линейной зависимости от производительности, вычисленной для прошлого проекта. Например, в рамках прошлого проекта была достигнута производительность в размере 1 строки программного кода за один рабочий час на программе в 5 KESLOC, следовательно, для разработки программы, состоящей из 50 KESLOC, при производительности 1 строка программного кода за один рабочий час, потребуется в 10 раз большее время. Это неверно, поскольку с увеличением размеров программы трудозатраты возрастают по экспоненциальному закону, а не по линейному.

**Таблица 12.1. Перевод конкретного числа функциональных баллов в тысячи строк KESLOC [26]**

| | A | B | C |
|---|---|---|---|
| 10 | Языки систем электронных таблиц | 6 | 50 |
| 11 | Языки запросов | 16 | 20 |
| 12 | SMALLTALK | 21 | 20 |
| 13 | OBJECTIVE-C | 26 | 12 |
| 14 | APL | 32 | 10 |
| 15 | STRATEGEM | 35 | 9 |
| 16 | База данных четвертого поколения | 40 | 8 |
| 17 | LOGO | 53 | 8 |
| 18 | BASIC | 64 | 5 |
| 19 | FORTH | 64 | 5 |
| 20 | LISP | 64 | 5 |
| 21 | PROLOG | 64 | 5 |
| 22 | Ada | 71 | 4,5 |
| 23 | MODULA-2 | 71 | 4,5 |
| 24 | PL/1 | 80 | 4 |
| 25 | RPG | 80 | 4 |
| 26 | Pascal | 81 | 3,5 |
| 27 | JOVIAL | 106 | 3 |
| 28 | FORTRAN | 106 | 3 |
| 29 | COBOL | 106 | 3 |
| 30 | CHILL | 106 | 3 |
| 31 | ALGOL | 106 | 3 |
| 32 | C | 150 | 2,5 |
| 33 | Макроассемблер | 213 | 1,5 |
| 34 | Ассемблер | 320 | 1 |

**B** — количество строк программного кода, соответствующее одному функциональному баллу

C — количество команд на выходном языке транслятора, соответствующее строке программного кода

Вывод, который непосредственно можно получить из этого исследования, заключается в том, что в условиях мелкомасштабных проектов каждый исполнитель знает все, что происходит ежедневно, а члены команды поддерживают друг друга, помогая решать сложные проблемы, разбираться в кодах и отлаживать модули. Ни у кого не возникает необходимости в документировании процесса, поэтому никто не задумывается над тем, чтобы упаковать данные измерений. В условиях небольших проектов каждый член группы вынужден брать на себя ответственность за судьбу всего проек-

та. Исполнитель в этом конкретном примере ощущает настоятельную необходимость иметь в своем распоряжении процесс оценки трудозатрат на тестирование для новых небольших проектов. Но в то же самое время, в состав предыдущих проектов не входили процессы сбора данных для базиса оценок.

При переходе от мелкомасштабных проектов к крупномасштабным возникает настоятельная потребность формализовать обмен данными, отчетность, руководящие действия, организационные роли, подготовку кадров и промышленные стандарты. Другими словами, возникает необходимость в определении и использовании полностью документированного процесса. В нескольких следующих главах будут показаны последовательности технологических операций процесса разработки современного программного обеспечения, при этом упор будет делаться на процесс тестирования и идентификацию его задач.

## Применение математических методов для оценки программного обеспечения

При моделировании процесса разработки программного обеспечения необходимо принимать во внимание тот факт, что различные модели жизненного цикла разработки имеют дело с современными моделями. В некоторых проектах определены технические требования, выявленные на начальной стадии проекта, и они остаются неизменными на протяжении всех этапов разработки. Минимальных затрат требует каскадный процесс разработки программного обеспечения. Каскадный (или "водопадный") процесс показан на рис. 12.1. Он получил свое название по аналогии с потоком самого процесса, Поскольку информационные потоки в диаграмме потоков данных направлены сверху вниз, подобно струям воды в водопаде. Стрелки, которые указывают в обратном направлении, суть потоки верификации. Они отвечают на вопросы наподобие: соответствует ли исходный код тому, чем он должен быть согласно детальной проектной документации.

Стадия тестирования, показанная на диаграмме каскадного процесса, включает комплексные, системные и приемочные испытания, в то время как стадия реализации, которая также показана на диаграмме, содержит задачи, относящиеся к тестированию модулей. Перед передачей программного кода независимым организациям, занимающимся тестированием, разработчики должны выполнить тестирование программных модулей. При решении задач на стадии тестирования каскадного процесса применяются технологии статического и динамического тестирования, рассмотренные в главах 3, 10 и 11.

Другой популярной моделью жизненного цикла программного обеспечения является модель, получившая название спиралевидного процесса, который показан на рис. 12.2. Этот процесс впервые был использован при разработке прототипов с целью повышения степени доверия к набору требований и обеспечения возможности экспериментирования с человеко-машинным интерфейсом, структурой базы данных, простотой использования, производительностью и удобством установки. Чтобы дать пользователям возможность поупражняться и получить от них отзывы, создаются инкрементальные сборки. Такие сборки являются должны обязательно тестироваться, однако слишком частые инкрементальные сборки иногда не позволяют группе тестирования полностью завершить прогон всего тестового набора.

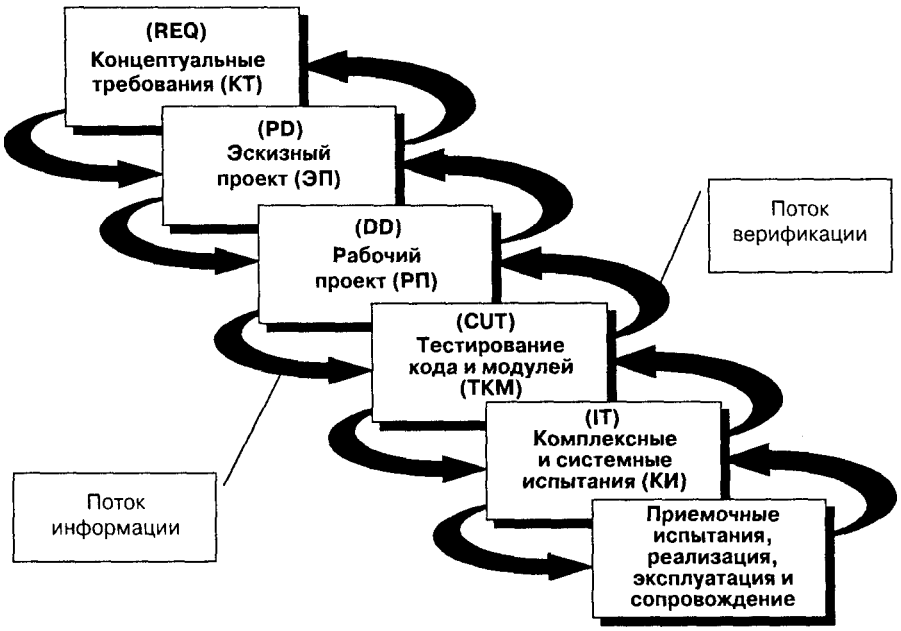



*Рис. 12.1. Каскадная модель*

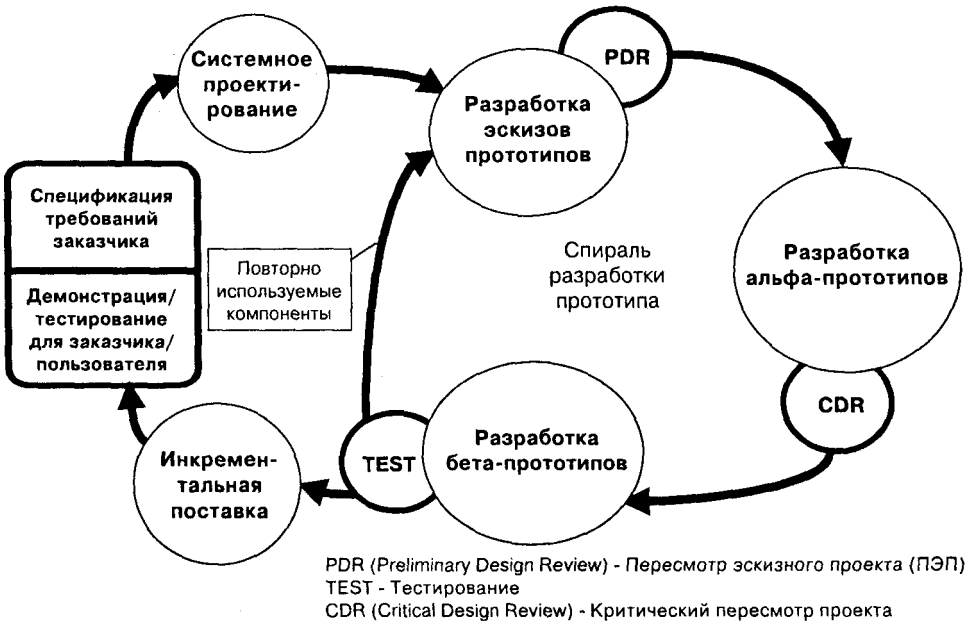



***Рис. 12.2. Спиралевидная модель***

В силу этого обстоятельства, задачей наивысшего приоритета становится обнаружение дефектов во вновь добавленной функциональности, но это, в свою очередь, приводит к спешке в разработке и реализации совершенно новых тестовых случаев, тестовых сценариев и ожидаемых результатов прогона тестов. Этот процесс набирает такие темпы, что становится хаотичным как для разработчиков, так и для тестировщиков. Если все этапы работ не будут тщательно выверены и распланированы, то по причине недостатка времени ни разработчики, ни тестировщики не успеют завершить работу и проанализировать ее результаты до появления следующей сборки.

Наилучшей моделью жизненного цикла разработки программного обеспечения, которое должно поступать на рынок с ограниченным набором функциональных возможностей с целью минимизации рисков, является спиралевидная модель. Спиралевидная модель обеспечивает плановый набор выпусков, благодаря чему переделка каких-либо пользовательских интерфейсов, структур баз данных или повышение производительности неоптимальных алгоритмов может выполняться в более поздних выпусках продукта.

Эта же модель поддерживает наилучший процесс в условиях, когда требования нечетко сформулированы либо отсутствуют прототипы продукта. Поставка предварительных сборок будущего программного продукта уменьшает риск неудачи всего проекта благодаря возможности заручиться поддержкой конечных пользователей на ранних стадиях разработки. Спиралевидная модель наилучшим образом подходит для проектов с изменчивыми требованиями, поскольку добавление, удаление и изменение требований можно встроить в следующие сборки, параллельно занимаясь сопровождением и разработкой новой сборки.

Вы, скорее всего, слышали часто употребляемый девиз достижения качества: "делай правильно с первой попытки". Если это относится к текущему типу проекта программного продукта, то наилучшим решением, требующим к тому же минимальных затрат, будет каскадная модель. Если пользователи не знают, чего они, в конце концов, хотят, или если может случиться так, что окончательная версия программного продукта когда-то в отдаленном будущем не удовлетворит ожидания заказчика, то наилучшим подходом к решению этой задачи, который минимизирует риск неудачи, будет спиралевидная модель.

В ранее рассмотренном жизненном цикле разработки программного обеспечения возможны изменения, обусловленные необходимостью привлечения внешних ресурсов субподрядчиков на проектирование, кодирование и тестирование модулей. На рис. 12.3 показана диаграмма шарнирно-каскадной модели, которая демонстрирует распределение ответственности в соответствии с промышленными стандартами между генеральным подрядчиком и субподрядчиками/партнерами.

Из приведенных примеров, иллюстрирующих широкое многообразие моделей жизненного цикла разработки, можно сделать вывод, что требования, предъявляемые к стоимостной модели, должны быть разбиты на блоки. Модель должна быть достаточно гибкой, чтобы делить сметную стоимость работ на стоимость работ, выполняемых генеральным подрядчиком, и стоимость работ, выполняемых субподрядчиком, равно как и в меру гибкой, чтобы выделить трудозатраты на тестирование, например, из затрат на управление разработкой и сопровождением программ или из затрат на кодирование. Это приводит к проекту, который рассматривается далее на примере изучения последовательности конкретных случаев.

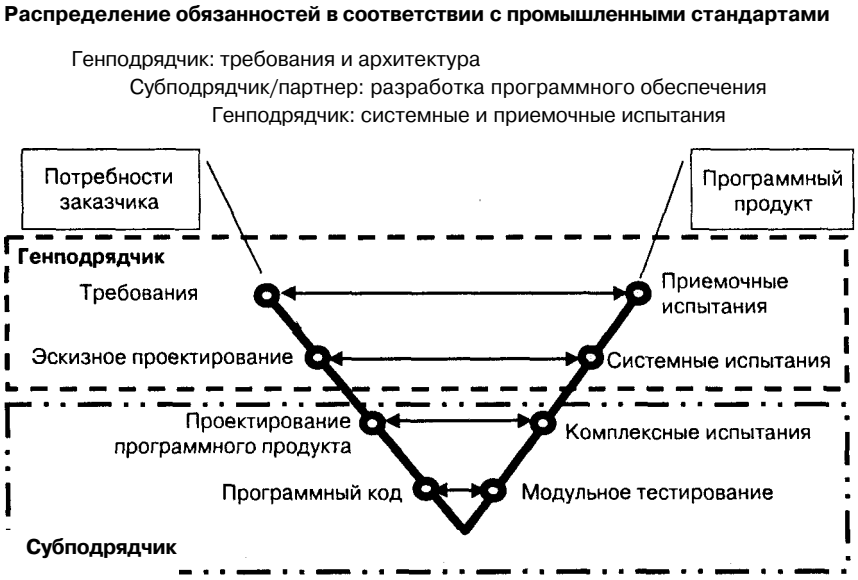



*Рис. 12.3. Шарнирно-каскадная модель, демонстрирующая распределение обязанностей между генеральным подрядчиком и субподрядчиком*

В [6] на примере 63 успешно выполненных проектов по разработке программного обеспечения выделяется три группы проектов, а именно: организованные, полуорганизованные и встроенные. Тем же, в [6], были разработаны три модели вычисления сметной стоимости: базовая, промежуточная и рабочая. В группу организованных проектов попадают те из них, в рамках которых работают небольшие группы, выполняющие хорошо Известную им работу, такую как, например, добавление сообщений в программе на языке COBOL. В группу встроенных проектов попадают проекты, в разработке которых участвовали группы с большим числом исполнителей и рисками, связанными с недостаточной осведомленностью в области программирования, недостаточными знаниями условий испытаний, нестабильностью требований, сокращенным графиком работ, отсутствием необходимых инструментальных средств, жесткими ограничениями по производительности и качеству. Группу полорганизованных проектов образуют проекты, которые находятся где-то посередине между группами встроенных и организованных проектов. На рис. 12.4, который представляет сводку уравнений модели COCOMO (Constructive Cost Model — конструктивная стоимостная модель), предложенную Барри Боемом (Barry Boehm), эти три группы называются режимами.

Для того чтобы упростить понимание различий между уравнениями организованных, полуорганизованных и встроенных режимов, на рис. 12.5 показаны графические диаграммы этих трех уравнений в тысячах строк KELOC (Equivalent Lines Of Code— эквивалентные строки кода). Диаграммы выглядят несколько запутанно, поскольку из них следует, что чем выше риск, связанный с программой, тем быстрее она завершится. По всей видимости, это можно объяснить тем, что руководство подбирает специалистов по разработке с таким расчетом, чтобы этот персонал смог ус-

пешно выполнять проекты с высокими рисками либо в условиях недостаточности ресурсов.

**Уравнения, используемые в базовой и промежуточной/детализированной моделях COCOMO**

**Базовая модель COCOMO**

$MM = 2{,}4\ (KLOC)^{1{,}05}$

$TDEV = 2{,}5\ (MM)^{0{,}38}$

***Промежуточная и детализированная модели COCOMO***

**Организованный режим**

$MM_{Adj} = (EAF)\ 3{,}2(KELOC)^{1{,}05}$

$TDEV = 2{,}5(MM\ до,)^{0{,}38}$

**Полуорганизованный режим**

$MM_{Adj} = (EAF)3{,}0(KELOC)^{1{,}12}$

$TDEV = 2{,}5(MMA_{dj})^{0{,}35}$

**Встроенный режим**

$MM_{Adj} = (EAF)2{,}8(KELOC)^{1{,}20}$

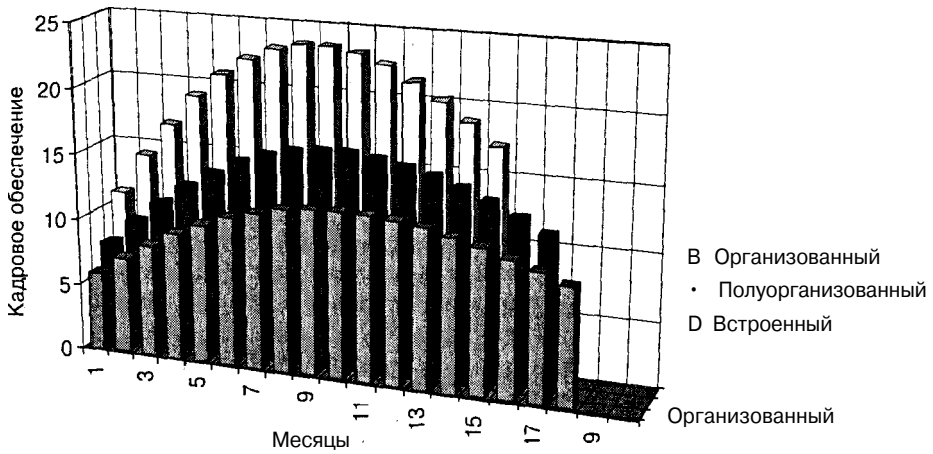
$TDEV = 2{,}5(MM_{Adj})^{0{,}32}$

**$MM_{м}$aint = (Интенсивность годичных изменений) $MM_{Adj}$ трудозатрат на эксплуатацию**

| | |
|---|---|
| ELOC | Эквивалентные строки программного кода |
| MM | Человеко-месяцы |
| EAF | Коэффициент уточнения трудозатрат |
| $MM_{Acj}$ | Скорректированные трудозатраты в человеко-месяцах |
| TDEV | Время на разработку |
| $MM_{MaJnt}$ | Годичные трудозатраты на эксплуатацию |

ELOC = (строки адаптированного программного кода) (1/100) (0,4*Процент модифицированных проектов + 0,3* Процент модифицированных программных кодов + 0,3 Процент модифицированных сборок)

***Рис. 12.4. Уравнения модели COCOMO для организованных, полуорганизованных и встроенных режимов***




**Разброс в показателях режимов COCOMO**

Относительно более медленный старт, начинающийся с меньших значений, и более высокие пиковые требования к кадровому обеспечению встроенного режима становятся очевидными на более высоких точках его графика, расположенного в глубине диаграммы; графики полуорганизованного и организованного режимов (на переднем плане) проходят ниже.

***Рис. 12.5. Режимы модели COCOMO: организованный, полуорганизованный и встроенный***

В промежуточных и детализированных моделях COCOMO применяется коэффициент EAF (Effort Adjustment Factor— коэффициент уточнения трудозатрат), представляющий собой произведение табличных значений драйверов стоимости программного обеспечения, сведенных в таблицу 12.2.

В базовой модели COCOMO коэффициент EAF (Effort Adjustment Factor— коэффициент уточнения трудозатрат) не используется. Единственное различие между промежуточной и детализированной моделями COCOMO заключается в том, что в промежуточной модели COCOMO на всех стадиях применяется один и тот же коэффициент EAF, в то время как в детализированной модели COCOMO перемножаются различные драйверы стоимости. В результате для каждой стадии получается свой коэффициент EAF, который применяется на соответствующем этапе вычисления трудозатрат. Единственное различие между моделью COCOMO и моделью REVIC (Revised and Improved COCOMO — пересмотренная и улучшенная модель COCOMO) связано с тем, какие коэффициенты и экспоненты присутствуют в уравнениях для вычисления количества человеко-месяцев (ММц) и времени разработки (TDEV).

На рис. 12.6 значение TDEV откладывается на временной оси графика; в рассматриваемом случае площадь, ограниченная 18 месяцами и кривой, представляющей изменения значений трудозатрат ($MM_{Adj}$), есть суммарное количество персонала за 18 месяцев. Главный драйвер стоимости модели COCOMO/REVIC — это число KLOC (Lines Of Code — тысячи строк программного кода), изображенное в центре диаграммы на рис. 12.6 в форме отверстия монетоприемника. Остальные драйверы стоимости показаны в виде циферблатов, стрелки которого указывают на избранный атрибут драйвера стоимости. В этом случае коэффициент EAF есть произведение значений, связанных (через поисковую таблицу) с каждой установкой циферблата. Подобное изображение, выполненное в духе графического пользовательского интерфейса, упрощает разработчику модели понимание последовательности событий этого процесса. Во-первых, необходимо установить показания циферблатов, затем опустить число KLOC в отверстие монетоприемника, запустить модель в работу и на выходе получить готовое произведение. График вычерчивает значения человеко-месяцев, при этом площадь под кривой представляет собой общие трудозатраты, которые потребуются на реализацию всех видов деятельности, которые необходимо выполнить при разработке программного продукта. Время на разработку (TDEV) откладывается по горизонтальной оси графика. Этот процесс повторяется для различных драйверов стоимости, достаточно ввести соответствующий драйвер либо ввести новые значения размеров и запустить модель.

Следует отметить, что существует прямая связь между трудозатратами ($MM_{Adj}$) и временем, затрачиваемым на разработку (TDEV), которая проявляется в том, что при уменьшении KLOC снижаются и трудозатраты, которые, в свою очередь, приводят к уменьшению времени до поставки. И наоборот, если составитель модели узнает, что работу над проектом прекращает специалист по ключевым системам проекта, который разрабатывает архитектуру и сам программный проект, необходимо скорректировать показания циферблатов ACAP и AEXP, вследствие чего трудозатраты ($MM_{Adj}$) могут возрасти, что, в свою очередь, приведет к увеличению времени на разработку (TDEV). И, наконец, если для вычислений используется средняя стоимость единицы персонала, занимающегося разработкой, то эта стоимость находится в линейной зависимости от вычисленных трудозатрат.

**Таблица 12.2. Значения драйвера стоимости после перемножения дают коэффициент уточнения трудозатрат [6].**

| | Рейтинги | | | | | |
|---|---|---|---|---|---|---|
| Драйвер стоимости | Очень низкий | Низкий | Номи-нальный | Высокий | Очень высок. | Исключ. высокий |
| Атрибуты программного продукта | | | | | | |
| RELY (Требуемая надежность программного продукта) | 0,75 | 0,88 | 1,00 | 1,15 | 1,40 | |
| DATA (Размер базы данных) | - | 0,94 | 1,00 | 1,08 | 1,16 | |
| CPLX (Сложность программного продукта) | 0,70 | 0,85 | 1,00 | 1,15 | 1,30 | 1,65 |
| Атрибуты компьютера | | | | | | |
| TIME (Ограничения на время выполнения) | | | 1,00 | 1,11 | 1,30 | 1,66 |
| STOR (Ограничения на основную память) | | | 1,00 | 1,06 | 1,21 | 1,56 |
| VIRT (Изменчивость виртуальной машинная) | | 0,87 | 1,00 | 1,15 | 1,30 | |
| TURN (Безремонтный срок службы компьютера) | | 0,87 | 1,00 | 1,07 | 1,15 | |
| Персональные атрибуты | | | | | | |
| ACAP (Производительность системного аналитика) | 1,46 | 1,19 | 1,00 | 0,86 | 0,71 | |
| AEXP (Квалификация системного аналитика) | 1,29 | 1,13 | 1,00 | 0,91 | 0,82 | |
| PCAP (Производительность программиста) | 1,42 | 1,17 | 1,00 | 0,86 | 0,70 | |
| VEXP (Опыт работы на виртуальной машине) | 1,21 | 1,10 | 1,00 | 0,95 | | |
| LEXP (Опыт работы с языками программирования) | 1,14 | 1,07 | 1,00 | 0,95 | | |
| **Атрибуты проекта** | | | | | | |
| MODP (Правила современного программирования) | 1,21 | 1,10 | 1,00 | 0,91 | 0,82 | |
| TOOL (Использование инструментальных средств программирования) | 1,21 | 1,10 | 1,00 | 0,91 | 0,83 | |
| SCED (Требуемый график разработки) | 1,23 | 1,08 | 1,00 | 1,04 | 1,10 | |

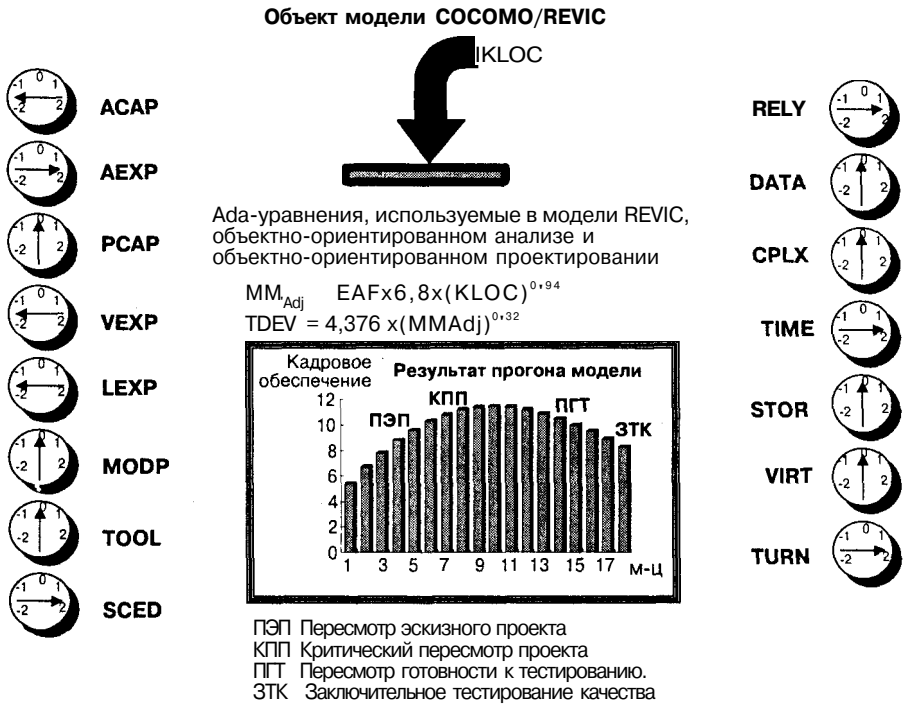



***Рис. 12.6. Графический пользовательский интерфейс модели COCOMO/REVIC** описывает **процесс вычислений***

Как отмечалось ранее, одно из требований, предъявляемых к модели сметной стоимости программного обеспечения, обусловливает ее модульность. Составитель такой модели, по меньшей мере, должен иметь возможность делать разрезы трудозатраты по стадиям разработки, по ролям разработчиков и по сборкам. На рис. 12.7 приводится уравнение рэлеевской кривой, а также трудозатраты (количество персонала) по месяцам, описываемые кривой кадрового обеспечения. Эта кривая обладает дополнительным свойством, которое заключается в том, что она разбита на типовые стадии разработки: REQ (документирование требований), PD (эскизное проектирование), DD (рабочее проектирование), CUT (тестирование кода и модулей), IT (комплексные и системные испытания). Вычисление сметной стоимости проекта невозможно, пока не будет завершена стадия REQ, поэтому в соответствии с соглашением о разметке оси абсцисс, первый месяц стадии PD помечается 1, в то время как метками двух месяцев, предшествующих стадии PD, когда выполняется REQ, будут 0 и -1. В случае спиралевидных процессов разработки на стадии REQ должны выполняться общие требования для всех сборок, таким образом, несложно обнаружить, что стадия REQ отделена от стадии PD на некоторый период задержки.

TDEV, время, затрачиваемое на разработку, и MM, человеко-месяцы, вычисляются по формулам, приведенным на рис. 12.6. Staffing(t) представляет собой количество персонала; значение этого показателя указывается на рис. 12.7 для каждого месяца, при этом t изменяется от **-1** до 19.

Кривые кадрового обеспечения модели COCOMO, взятые из динамической электронной таблицы модели COCOMO

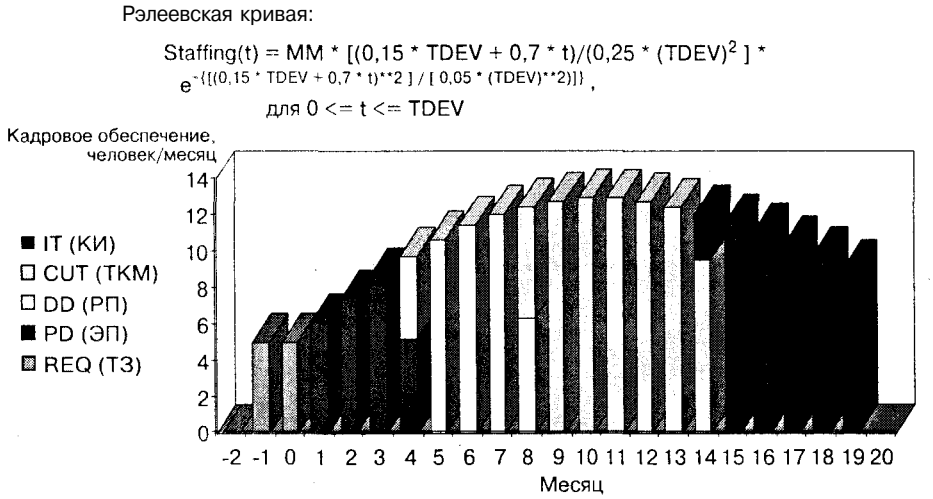



*Рис. 12.7. Графическая схема модели COCOMO трудозатрат на стадии разработки*

И хотя модель COCOMO не предназначена для получения затрат на эксплуатацию для унаследованных систем, вариант, показанный на рис. 12.8, обеспечивает достаточно точные результаты. Затраты на эксплуатацию $MM_{Maint}$ в течение трех лет после выпуска унаследованной системы могут быть подсчитаны на основе интенсивности годичных изменений, которая согласно эмпирическому правилу, составляет в среднем, соответственно, 15%, 10% и 5% от трудозатрат на создание унаследованной системы. Пример, представленный на рис. 12.8, построен для программы объемом в 50 KLOC, на разработку которой потребовалось 362 человеко-месяца, при этом в течение первых 12 месяцев 4,525 человека были заняты поддержкой унаследованной системы. Такой подход можно применить в случае, когда подрядчик получает новый программный COTS-продукт (commercial-off-the-shelf, доступный в продаже) такого же объема (50 KLOC).

Предположим, что подрядчик настаивает на добавлении новых функциональных свойств в COTS-продукт, на осуществление которого потребуется не менее двух лет. Этот подрядчик должен требовать новых разработок объемом X KLOC, интегрированных в сборку размером X + 50 KLOC, и добавить к соответствующим трудозатратам 4,525 человеко-месяца на первый год и 3,017 человеко-месяца на второй год к трудозатратам на эксплуатацию COTS-продукта. Имея среднюю оплату $55 за человеко-час, или $8305 за человеко-месяц, примите, что один месяц состоит из 151 оплачиваемого часа. Стоимость трудозатрат за два года составит в этом случае $8305 * 7,542 = $62636. Обратите внимание на то обстоятельство, что в данном случае ничего не сказано по поводу того, что какие средства были потрачены на унаследованный COTS-продукт его поставщиком, поскольку значение 362 человеко-месяца было вычислено на основании коэффициента EAF вашей компании, который определялся с учетом особенностей процесса разработки программного обеспечения внутри компании.

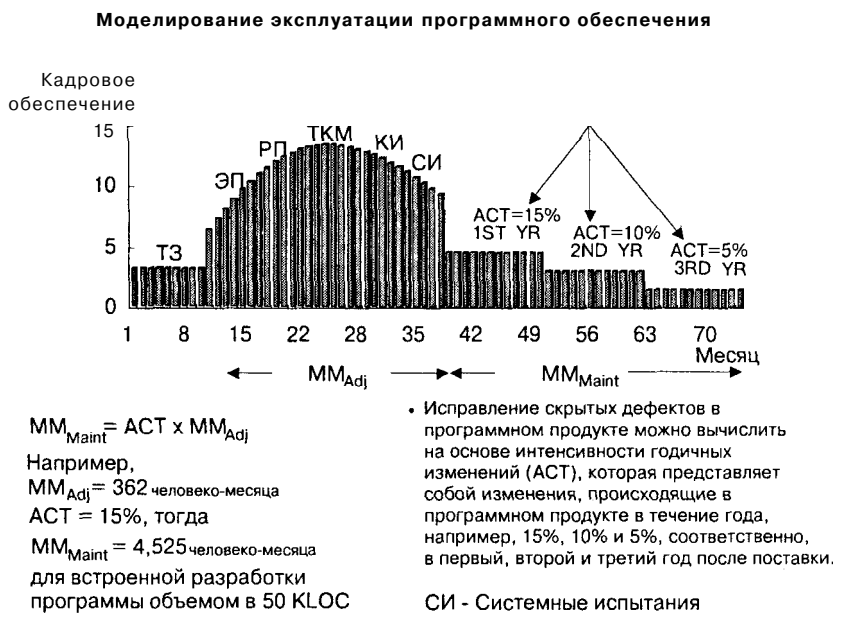



*Рис. 12.8. Уровень трудозатрат на эксплуатацию, подсчитанный на основе измерения площади фигуры между осью абсцисс и кривой кадрового обеспечения*

Предположим, стало известно, что конкуренту вашей компании потребовалось четыре года, чтобы завершить COTS-продукт объемом в 50 KLOC. В соответствие с контрактом, который заключила ваша компания, этот продукт является основой для внесения усовершенствований. Несут ли эти сведения информацию, которая могла бы укрепить ваши позиции в конкурентной борьбе? Имея в своем распоряжении уравнения математической модели, можно восстановить значение

($MM_{Adj}$) из уравнения для TDEV. Далее, имея значение ($MM_{Adj}$) и учитывая то, что объем COTS-продукта равен 50 KLOC, единственной неизвестной величиной в уравнении для ($MM_{Adj}$) остается коэффициент EAF. Зная величину коэффициент EAF для COTS-продукта у конкурента, можно определить его затраты на выполнение следующего проекта, скажем, объемом в 75 KLOC. Постоянно совершенствуя драйверы стоимости, которые используются для вычисления значений коэффициент EAF, ваша компания сможет победить в конкурентной борьбе, создавая программные продукты за более низкую цену. Последовательно управляя математической моделью, ваша компания в конечном итоге с полным правом сможет сказать: "Мы можем поставлять программный продукт в более сжатые сроки и за меньшую цену".

### ПРИМЕР 2. ПОСТРОЕНИЕ МОДЕЛИ COCOREV (ГЭРИ КОББ)

Математическое моделирование жизненного цикла программного обеспечения всегда вызывало у меня повышенный интерес. Следствием такой заинтересованности стали краткий обзор различных случаев использования моделей составления смет, которые мне удалось обнаружить, и исследование гибкости этих моделей. С самого-начала я отдал предпочтение структуре электронной таблицы, поскольку она обладала модульностью и графическим интерфейсом, необходимыми для достижения поставленных целей. Я принял решение написать для наиболее сложных случаев программу детализированной модели COCOMO, в которой на каждой стадии применяются собственные коэффициенты EAF. В модели COCOREV используются параметризованные коэффициенты и экспоненты для вычисления значений ($MM_A$dj) и TDEV, благодаря чему одна и та же электрон--ная таблица применяется для решения уравнений модели REVIC. Мы решили назвать эту модель, в основу которой положена электронная- таблица, табличной моделью COCOREV. Модель COCOREV предоставит примеры использования математической модели для составления сметы затрат на разработку программного обеспечения. В равной степени найдут применение и уравнения для полуорганизованных проектов (режим 2), полученные Барри Боемом, а в промежуточной модели вычисления сметной стоимости принимается, что для всех стадий разработки используется одно и то же значение коэффициента EAF, равное 0,5.

В рассматриваемом конкретном примере можно выделить три части:

- Спиралевидный жизненный цикл с пятью сборками объемом 15 KLOC, которые интегрируются в программный код существующей системы и создаются последовательно.
- Каскадный жизненный цикл с одной сборкой объемом 75 KLOC, создаваемой один раз.
- Многоярусный спиральный жизненный цикл с пятью сборками объемом 15 KLOC, которые интегрируются в программный код существующей системы, при этом спирали создаются каждые два месяца.

С итоговыми результатами этого анализа, проводимого с целью выбора компромиссного решения, можно будет познакомиться после того, как будут представлены данные третьей части. На рис. 12.9 показаны допущения, принятые для первой части примера.

Модель COCOREV представляет собой рабочий журнал, содержащий некоторое множество рабочих страниц. Главная страница, именуемая Титульным листом, содержит все входные данные модели и сводку всех выходных результатов пяти страниц, на которых уравнения модели COCOREV применяются к индивидуальным сборкам, помеченным как Сборка 1, Сборка 2, Сборка 3, Сборка 4 и Сборка 5. Еще одна рабочая страница, помеченная как Таблицы, содержит фиксированные константы рассматриваемой модели.

## Пример - Часть 1

- Допущения
  - 5 сборок
  - 15 KDSI каждая
  - Каждая сборка интегрируется с промежуточными результатами предыдущих сборок
  - Для реализации каждой сборки используется язык программирования **C**++
  - В рамках одной сборки подробно рассматриваются:
    - Планирование тестирования
    - Верификация и аттестация

*Рис. 12.9. Список допущений для части 1 примера*

На рис. 12.10 показан блок входных данных Титульного листа, подробно описывающий каждую из сборок объемом в 15 KELOC. Сборка интегрируется сама с собой™ с предыдущими унаследованными сборками и реализуется на языке C++ со значением коэффициента EAF, равным 0,5, для всех стадий разработки. Мы выбираем режим 2 которому соответствует полуорганизованная модель COCOMO. Столбец KDSI (thousands of delivered instructions — тысячи поставленных инструкций) Титульного листа заполняется значениями, которые начинаются со сборки объемом в 15 KDSI и с каждой строкой возрастают на 15 KDSI. Задержка начала работ в начале равна 0, а затем возрастает на 10 месяцев с каждой сборкой. Это делается для того, чтобы показать, что на разработку каждой сборки будет затрачиваться не более 10 месяцев.

**Модель COCOREV Версия 1.02**

**Разработанный программный продукт**

| Номер сборки | Язык программирования | KELOC | KDSI | Задержка начала работ |
|---|---|---|---|---|
| Сборка 1 | C++ | 15,000 | 15,000 | 0,0 |
| Сборка 2 | C++ | 15,000 | 30,000 | 10,0 |
| Сборка 3 | C++ | 15,000 | 45,000 | 20,0 |
| Сборка 4 | C++ | 15,000 | 60,000 | 40,0 |
| Сборка 5 | C++ | 15,000 | 75,000 | 50,0 |

**Разработанный программный продукт**

| Номер сборки | EAF:REQ | EAF:PO | EAF:DD | EAF:CUT | EAF: IT |
|---|---|---|---|---|---|
| Сборка 1 | 0,500 | 0,500 | 0,500 | 0,500 | 0,500 |
| Сборка 2 | 0,500 | 0,500 | 0,500 | 0,500 | 0,500 |
| Сборка 3 | 0,500 | 0,500 | 0,500 | 0,500 | 0,500 |
| Сборка 4 | 0,500 | 0,500 | 0,500 | 0,500 | 0,500 |
| Сборка 5 | 0,500 | 0,500 | 0,500 | 0,500 | 0,500 |

| | |
|---|---|
| KELOC | Тысячи эквивалентных срок программного кода |
| KDSI | Тысячи инструкций поставленного исходного кода |
| Задержка начала работ | Число месяцев перед началом разработок |
| EAF | Коэффициент уточнения трудозатрат |

*Рис. 12.10. Состояние входных значений Титульного листа .модели COCOREV (название сборки, язык программирования, задержка начала работ и коэффициенты EAF по стадиям и по сборкам)*

Рис. 12.11 является частью комплексного вывода на Титульном листе модели COCOREV Поскольку с началом работ над сборкой 1 задержки не было, они начинаются на месяце 1, в то время как работы над сборкой 2 были начаты с задержкой на 10 месяцев, т.е. они начнутся в течение месяца 11 и т.д. Стадия описания требований (REQ) для всех пяти сборок происходит в течение месяцев -1 и 0, а все пять сборок будут завершены до наступления месяца 50. В то время как пики всех пяти кривых кадрового обеспечения появляются в начале фазы CUT, несложно заметить, что пики кадрового обеспечения монотонно возрастают от сборки 1 до сборки 5, что объясняется объемами интегрируемых программных модулей и тестовых работ, возрастающих с увеличением KSDI короче говоря, это отнюдь не пять идентичных сборок, выполняемых последовательно. Если вас интересует, "в какую сумму обойдутся четыре года разработки программного обеспечения , то ответом будет $1,593 миллиона, если предположить, что средняя ча-

совая оплата труда разработчиков останется на уровне $55 на протяжении всего четырехлетнего периода работы. Это еще один ответ, который можно найти на Титульном листе. Итак, в сводке Титульного листа появляется итоговое значение 28968 человеке*-часов, разбитое по стадиям и представленное в человеко-месяцах следующим образом: REQ = 14,83, PD = 32,50, DD = 41,92, CUT = 60,54, IT = 42,05, что в сумме составляет 191,94 человеко-месяца.

В центре внимания рис. 12.12 находится только распределение ресурсов во время создания сборки 1. Обратите внимание на то обстоятельство, что REQ = 2,75 человеко-месяца и что это составляет справедливую долю от общего значения REQ = 14,83 человеко-месяца, затраченных на документирование всего продукта со множеством сборок. Общая стоимость разработки после формулирования требований составляет 34,18 минус 2,75, затраченных на REQ, что дает 31,43 человеко-месяца, и это значение совпадает с общим значением, указанным в правом столбце. Из графика загрузки персонала, полученного с помощью модели COCOREV, следует, что преимущество программного обеспечения с точки зрения используемого персонала в этом проекте должно возрасти с 2 исполнителей до 4,5 при пике нагрузки и снизиться до 1,5 на протяжении последнего месяца.

Аналог сборки 1, а именно, сборка 2, представлен на рис. 12.13. В чем состоят различия между сборкой 1 и сборкой 2? Итак, во-первых, прежде чем сборку 2 можно будет поставлять, она должна быть интегрирована со сборкой 1, после чего обе сборки должны быть подвергнуты совместному тестированию. Второе отличие заключается в том, что разработка сборки 2 задержана на 10 месяцев. Это означает, что ее разработка начинается на 11-м месяце, т.е. по истечении 1,5 месяцев после выпуска сборки 1. Другими словами, требования, предъявляемые к сборке 2, ждут своего часа около 10 месяцев, а за это время требования вполне могут претерпеть определенные изменения, причем совершенно бесплатно, пока кто-нибудь не возьмет на себя обязанности сопровождения документов с формулировками требований. В данном случае выражение "совершенно бесплатно" является доказательством того, что приверженцы спиралевидного-подхода обычно принимают меры в поддержку своего утверждения о том, что спиралевидный процесс уменьшает риски самопроизвольных изменений требований. В силу каких причин затраты на требования в случае сборки 2 выше, чем затраты на требования в случае сборки 1 (соответственно, 2,9 и *2,7* человеко-месяца)? Это объясняется более - высокой сложностью требований сборки 2 к интегрированию со сборкой 1.

Из соображений краткости мы не будем показывать аналогичные графики для сборок 3 и 4. На рис. 12.14 представлена кривая кадрового обеспечения для последней из пяти сборок. Сравнивая эту кривую кадрового обеспечения с аналогичной кривой для сборки 1, можно отметить, что суммарные трудозатраты, за вычетом затрат на документирование требований, составляют 38,22 - 31,33 = 6,89 человеко-месяцев. Для проекта, на разработку которого уходит немногим более 9 месяцев, для создания сборки 5 требуется на одного исполнителя больше (количество персонала), чем для сборки 1. Опять-таки, эти расчеты подтверждают тот факт, что наличие в случае сборки 5 унаследованного кода достаточно большого объема обусловливает большие затраты времени и усилий по сравнению со сборкой 1, у которой вообще нет унаследованных кодов.

В начале этой главы мы поставили перед собой задачу дать по возможности более точное описание способа прогнозирования трудозатрат на подготовку и выполнение тестовых работ с разбивкой по месяцам, которое обычно называется верификацией и аттестацией (V6iV). После предварительного описания возможностей модели COCOREV мЫ продолжим рассмотрение примера с последовательностью из пяти сборок, уделяя основное внимание упомянутым двум тестовым функциям. Обратимся снова к сборке 1 и разделим работы по тестированию на две категории: планирование испытаний-(рис.12.15) и верификация и аттестация (рис.12.16). Эти данные содержатся в электронной таблице сборки 1 в рабочем журнале модели COCOREV. Термин *персонал тестирования (test staff)* означает совокупность персонала, составляющего планы проведения испытаний, и персонала, выполняющего верификацию и аттестацию, например, совокупное число исполнителей, представленных на графиках, изображенных на рис. 12.15 и 12.16. .


*Рис.12.11. Титульный лист модели COCOREV, на котором показаны пять инкрементально увеличивающихся сборок, которые разрабатываются после выполнения общей для всех сборок стадии документирования требований*

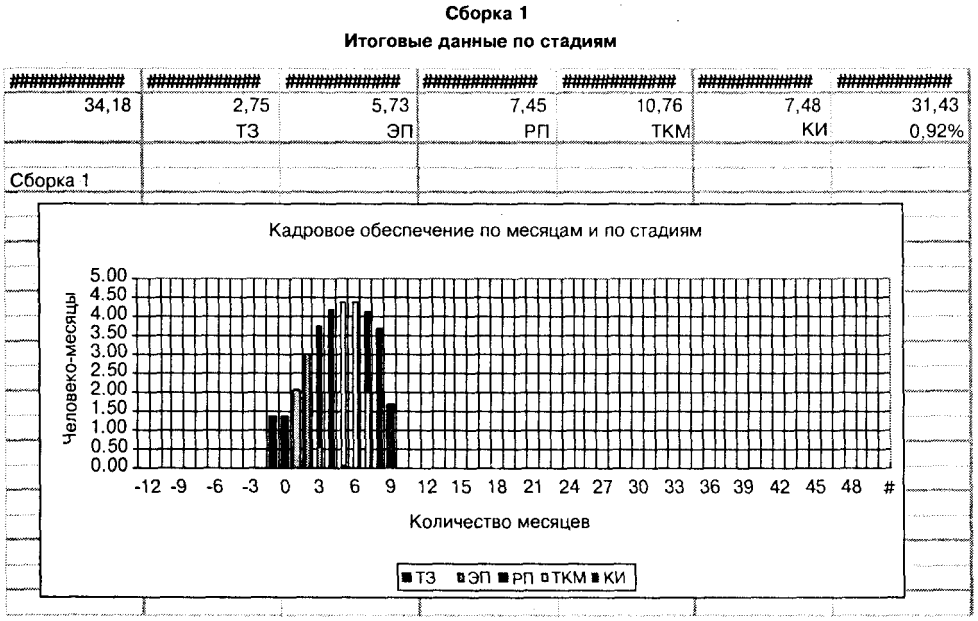



*Рис.12.12. Электронная таблица для сборки 1 согласно модели COCOREV выделяет информацию, специфическую для сборки 1*

**Сборка 2**

**Итоговые данные по стадиям**

| ########## | ########## | ########## | ########## | ########## | ########## | ########## |
|---|---|---|---|---|---|---|
| 37,10 | 2,90 | 6,27 | 8,10 | 11,70 | 8,13 | 34,20 |
| | ТЗ | ЭП | РП | ТКМ | КИ | 1,06% |

Сборка 2

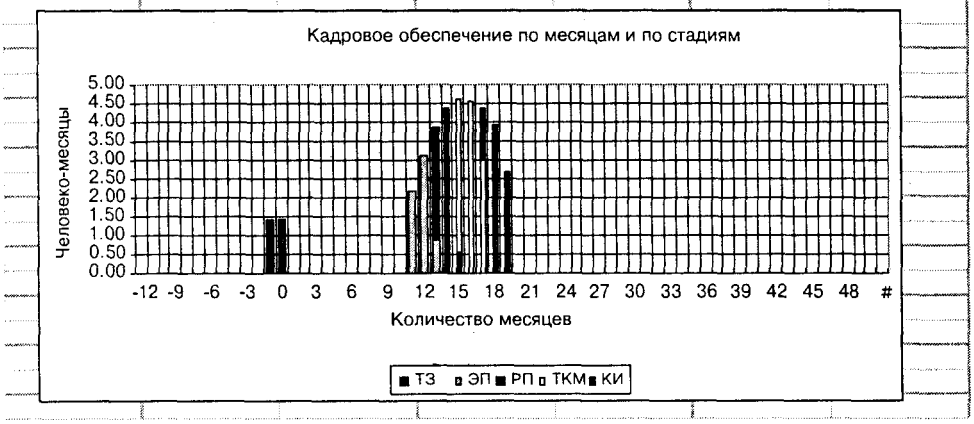



*Рис.12.13. Электронная таблица для сборки 2 согласно модели COCOREV выделяет информацию, специфическую для сборки 2*

**Сборка 5**

**Итоговые данные по стадиям**

| ########## | ########## | ########## | ########## | ########## | ########## | ########## |
|---|---|---|---|---|---|---|
| 41,34 | 3,12 | 7,05 | 9,06 | 13,07 | 9,05 | 38,22 |
| | ТЗ | ЭП | РП | ТКМ | КИ | 1,19% |

Сборка 5


*Рис.12.14. Электронная таблица для сборки 5 согласно модели COCOREV выделяет информацию, специфическую для сборки 5*


*Рис.12.15. Кривая кадрового обеспечения планирования проведения испытаний сборки 1 согласно модели COCOREV*

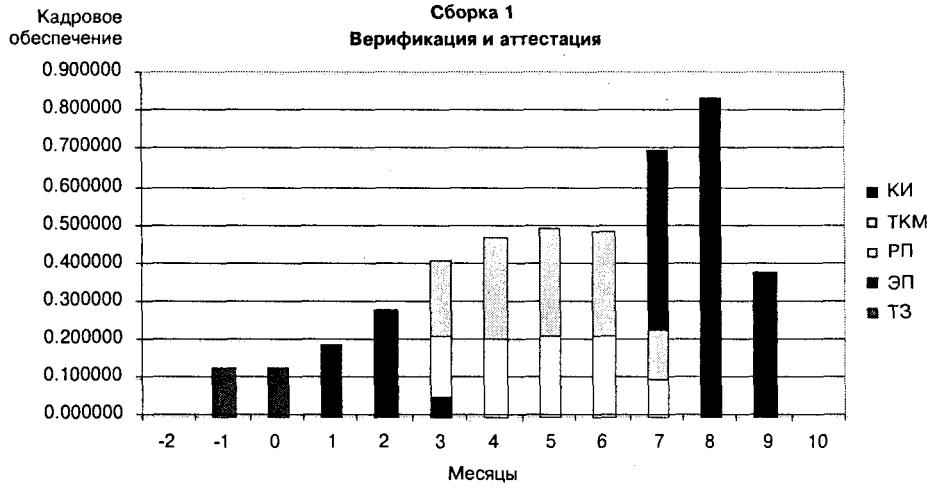



***Рис.12.16. Кривая кадрового обеспечения планирования проведения верификации и аттестации сборки 1 согласно модели COCOREV***

**Что собой представляет персонал, включенный в состав группы, формулирующей технические требования? Модель COCOREV выделяет персоналу, проводящему тестирование, некоторое количество часов для участия в документировании требований на протяжении месяцев -1 и 0, предшествующих началу разработки сборки 1. Персонал тестирования требует от авторов требований дать пояснения, что означают эти требования применительно к тестированию каждой важной функции, создавая тем самым основу для составления персоналом тестирования соответствующих планов проведения испытаний, тестовых случаев и результатов прогона тестов. На некоторых стадиях разработчикам полезно также время от времени освежать в памяти технические требования на проведение тестирования, чтобы можно было включать различные специализированные тестовые процедуры в проекты и далее встраивать их *в* программное обеспечение.**

Примерами специализированных тестовых процедур могут служить регистрация транзакций, откат всех транзакций баз данных, ведение файлов предыстории, в которых накапливается хронология изменения данных, последовательность прохода ветвей кода или выхода без заказа из пользовательских корзин на Web-сайтах электронной коммерции. Исполнители из числа персонала тестирования с их заряженностью на обнаружение дефектов, должны ознакомиться с эскизной и вспомогательной документацией, чтобы обнаружить и задокументировать дефекты в требованиях. В конечном итоге, это наиболее подходящее место для экономически эффективного отлавливания дефектов — когда при наличии всех мощных аппаратно-программных средств для устранения всех этих дефектов достаточно будет только карандаша и резинки.

При сравнении приведенных выше графиков несложно заметить, что в качестве стадий в обоих выбраны одни и те же месяцы, и что анализ ресурсов благоприятствует планированию на ранних стадиях, а верификацию и аттестацию (V&V) лучше выполнять позже, на стадии комплексных испытаний (IT). Это не расходится с предназначением каждой задачи, например, планирование обнаружения дефектов во время использования технологий статического тестирования на стадиях REQ, PD, DD и CUT с последующим выполнением планов проведения испытаний, обнаружением и документированием дефектов с использованием динамических технологий тестирования на стадиях IT и приемочных испытаний. Модель COCOREV не поддерживает точку зрения некоторых руководителей, которые утверждают, что после того, как разработчики построят готовый к запуску программный продукт, можно подключать к работе независимую команду тестирования, поставив перед ней задачу отыскания всех дефектов. Напротив, удачными считаются такие проекты, во время разработки которых планирование тестовых работ и прогон тестов осуществляются на всех стадиях. Внимательность и осторожность при обнаружении и исправлении дефектов с использованием статических средств на первых трех стаднях разработки — вот на чем делается акцент в технологии быстрого тестирования.

Допущения анализируемого примера 2 представлены на рис.12.17. Они мало чем отличаются от допущений примера 1, однако на этот раз мы намерены изменить одно важное условие для исследования отношения риски/выигрыш, а именно, выбрать в качестве жизненного цикла вместо спиральной каскадную модель. Для этого поставим перед собой задачу объединить все 75 KLOC в сборку 1 и интегрировать ее саму с собой (75 K.DSI), разумеется, отказавшись от использования четырех остальных сборок модели COCOREV.

После применения модели COCOREV на сборке 1 были получены следующие результаты, разбитые по стадиям и выраженные в человеко-месяцах: REQ = 11,84, PO = 38,50, DD = 42,85, CUT = 58,86, IT = 51,54, что в совокупности дает 204,59 человеко-месяца. Это составляет 30983,7 человеко-часов, учитывая, что месяц приравнивается к 151 оплачиваемому человеко-часу. В пересчете на денежные единицы получается $1,699 милпиона, исходя из предположения, что стоимость человеко-часа составляет $55. График, изображенный на рис.12.18, показывает, что на разработку проекта потребуется более 16 месяцев, при этом работы над проектом начинаются после трехмесячного периода формулирования требований. Пик кривой кадрового обеспечения приходится на стадию CUT на уровне 15 человек в течение трех месяцев.

## Пример - Часть 2

- **Допущения**
  - **1 сборка**
  - 75 KDSI
  - **Для реализации каждой сборки используется язык программирования C++**
  - **В рамках одной сборки подробно рассматриваются:**
    - **Планирование тестирования**
    - **Верификация и аттестация**

*Рис. 12.17. Список допущений для части 2 примера*

**На рис.12.19 и 12.20 показаны, соответственно, кривые трудозатрат на составление планов проведения испытаний и на верификацию и аттестацию. Большая часть трудозатрат на планы проведения испытаний приходится на стадии PD, DD и CUT, при этом в течение девяти следующих подряд месяцев потребуется 0,8 человека каждый месяц. Наибольшее количество персонала, занимающегося верификацией и аттестацией, приходится на стадию IT и составляет 2,5 человека в течение трех месяцев подряд.**

**Этим заканчивается изучение части 2 примера. На рис. 12.21 представлен список допущений для части 3 данного примера.**


*Рис.12.18. Каскадная модель COCOREV выполнения проекта в 75 KLOC на языке C++*


*Рис.12.19. Кривая кадрового обеспечения планирования тестовых работ для каскадной модели выполнения проекта в 75 KLOC*


*Рис. 12.20. Кривая кадрового обеспечения верификации и аттестации для каскадной модели выполнения проекта в 75 KLOC*

## Пример - Часть 3

- Допущения
  - 5 сборок, выполняемых каждые 2 месяца
  - 15 KDSI каждая
  - Каждая новая сборка интегрируется с результатами предыдущих сборок
  - Для реализации каждой сборки используется язык программирования C++

***Рис.12.21. Список допущений для части 3 примера***

Результаты этой части иллюстрируются графиком, который показан на рис. 12.22. Кадровое обеспечение каждой стадии выглядит следующим образом: REQ = 14,83, PD = 32,50, DD = 41,94, CUT = 60,54, IT = 42,05, что в совокупности дает 191,86 человеко-месяца. Это составляет 28971,7 человеко-часов. Полагая, что месяц содержит 151 оплачиваемый 151 час, то в пересчете на денежные единицы получается $1,593 миллиона при стоимости человеко-часа $55.

Поскольку на каждой стадии разработки программного продукта, а именно, на стадиях PD, DD, CUT, IT, трудится персонал, прошедший специальную подготовку и получивший сертификат на выполнение всех необходимых задач, конкретное значение многоярусного спиралевидного жизненного цикла состоит в том, что персонал может завершить соответствующую стадию на одной сборке и перейти к выполнению той же стадии на следующей сборке. При этом каждый исполнитель будет непрерывно занят решением своей задачи на протяжении года. Другое важное значение многоярусного спиралевидного жизненного цикла связано с тем, что пользователи начинают работать с первой частью новой системы примерно на полпути многоярусного спиралевидного жизненного цикла и, регулярно продолжают получать выпуски с дополнительной функциональностью на про-, тяжении остальных 10 месяцев жизненного цикла.

С точки зрения персонала, составляющего планы проведения испытаний и выполняющего верификацию и аттестацию, получение нового выпуска можно спланировать к концу каждого двухмесячного периода из десяти последних месяцев жизненного цикла.

С. гЬинзнсовон точки з"вния по"ход с использованием спи"эл€5.и"ной модели жизненно-го цикла обладает определенными преимуществами перед подходом с применением каскадной модели. В таблицу 12.3 сведены самые важные статистические данные каждой из частей рассматриваемого примера.

Наиболее эффективной моделью для анализируемого примера является многоярусный спиралевидный жизненный цикл, поскольку он обеспечивает более раннюю передачу результирующего приложения конечным пользователям, определяет необходимый работочий персонал для каждой стадии приблизительно на год {причем количество сотрудников в течение года не изменяется), характеризуется меньшими расходами по сравнению С каскадной моделью и дает возможность воспользоваться преимуществами от совершенствования процесса в каждой стадии по мере продвижения через все пять сборок.

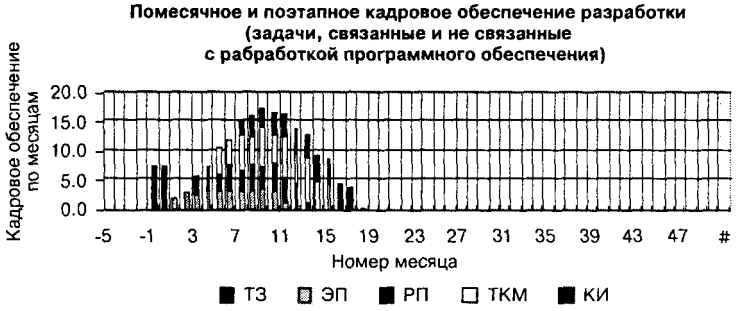



*Рис.12.22. Многоярусный спиралевидный жизненный цикл*

**Таблица 12.3. Компромиссная матрица для примера, состоящего из трех частей**

| Часть | Описание | Затраты в миллионах долларов | Максимальное количество персонала | TDEV |
|---|---|---|---|---|
| 1 | 5 сборок, 15KLOC, попарная интеграция | 1,593 | 5 | 49 |
| 2 | 1 сборка, 75 KLOC | 1,699 | 14 | 15 |
| 3 | 5 сборок, 15KLOC, накопительная интеграция | 1,593 | 18 | 17 |

# Технология функциональных баллов

**В** [1] предложена методология классификации программного обеспечения по размерам, которая построена на предположении, что производительность разработки программного обеспечения каким-то образом связана с функциональными возможностями. Области функциональных возможностей, которые при этом учитываются, называются значениями информационного домена, часть из которых перечислена ниже:

- Количество вводов пользователя — подсчет числа вводов пользователя. Учитывается каждый ввод пользователя, который представляет пользователю отличающуюся информацию, связанную с приложением. Вводы должны отделяться от запросов, которые учитываются отдельно.

- Количество выводов для пользователя — подсчет числа выводов для пользователя. Учитывается каждый вывод для пользователя, предоставляющий отличающуюся информацию из приложения. В этом контексте под выводами понимаются отчеты, экраны, сообщения об ошибках и тому подобное. Индивидуальные элементы данных внутри отчетов отдельно не подсчитываются.
- Количество запросов пользователя — запрос определяется как интерактивный ввод, вызывающий ту или иную немедленную реакцию программного обеспечения в форме интерактивного вывода.
- Количество файлов — учитывается каждый логический главный файл (т.е. логически сгруппированные данные, которые могут быть частью крупной базы данных, состоящей из отдельных файлов).
- Количество внешних интерфейсов — учитываются все машиночитаемые интерфейсы (например, файлы данных на некоторых носителях, используемых для передачи в другие системы).

Введите полученные числовые значения в столбец "Количество" таблицы, изображенной на рис. 12.24.

На рис. 12.23 представлен первый этап двухэтапного процесса подсчета функциональных баллов (function points, FP). Этап 1 начинает исполнитель, имеющий опыт оценки методом функциональных баллов. Исполнитель должен ответить на вопросы специальной анкеты, используя при этом числа, указанные в заголовке анкеты. После получения ответов на все 14 вопросов анкеты, необходимо сложить все числа, представляющие ответы, и передать полученную сумму на следующий шаг.

На шаге 2 потребуется перемножить эти числа на соответствующие веса, которые указаны в столбцах, помеченных как "Простой", "Средний" и "Сложный", просуммировать полученные произведения и поместить сумму в поле "Общая сумма" в соответствии с инструкциями, приведенными в начале раздела. И, наконец, воспользуйтесь формулой, показанной в нижней части рис. 12.24. В качестве сомножителя $SUM(F_i)$ необходимо подставить сумму коэффициентов уточнения сложности, полученных на шаге 1.

Функциональный балл представляет собой взвешенную метрику индексного типа. предназначенную для изменения объема функциональности программного пакета, каким его видит конечный пользователь. В начале семидесятых, работая в компании IBM, Аллан Альбрехт (Allan Albrecht) ввел понятие функционального балла. Он полагал, что функциональные возможности программного приложения представляет собой более важную характеристику размера программного обеспечения, нежели число LOC (Lines Of Code — строк программного кода), которое было тогда традиционной единицей измерения объема программного продукта. Функциональные баллы ставятся в один ряд с функциями обработки информации, которые поставляются пользователям. Функциональные баллы представляют собой средство измерения размеров программного продукта, которые не зависят от методов разработки, технологий и языков программирования. Как метрика размеров программного обеспечения, значения в функциональных баллах обладают большей надежностью и могут определяться на ранних стадиях жизненного цикла разработки программного продукта.

Шаг 1

- **С каждым вопросом связан коэффициент уточнения сложности Fi, где i = 1,..., 14. Дайте оценку каждого коэффициента по шкале от 0 до 5.**

(0 = влияние отсутствует; 1 = эпизодическое влияние; 2 = умеренное влияние; 3 = среднее влияние, 4 = значительное влияние; 5 = существенное влияние)

1. Требуются ли в системе механизмы резервирования и восстановления?
2. Требуется ли обеспечить обмен данными?
3. Являются ли рассматриваемые функции функциями распределенной обработки данных?
4. Являются ли производительность критическим фактором?
5. Будет ли система работать в условиях существующей, интенсивно используемой операционной системы?
6. Требуется ли в системе интерактивный ввод данных?
7. Требует ли интерактивный ввод данных заполнения транзакции на нескольких рабочих экранах?
8. Производится ли обновление главных файлов в интерактивном режиме?
9. Относятся ли вводы, выводы, файлы или запросы к категории сложных?
10. Относится ли внутренняя обработка к категории сложной?
11. Относится ли разрабатываемый код к категории используемого многократно?
12. Включены ли в проект преобразование и установка?
13. Предназначена ли система для многократной установки в различных организациях?
14. Разрабатывалось ли приложение с целью упростить изменения и облегчить работу пользователя?

*Рис. 12.23. Коэффициенты уточнения сложности в функциональных баллах*

**Шаг 2**

| Параметр измерения | Количество | Весовые коэффициенты: Простой | Средний | Сложный | |
|---|---|---|---|---|---|
| Количество вводов пользователя | | x 3 | 4 | 6 = | |
| Количество выводов для пользователя | | x 4 | 5 | 7 = | |
| Количество запросов пользователя | | x 3 | 4 | 6 = | |
| Количество файлов | | x 7 | 10 | 15 = | |
| Количество внешних интерфейсов | | x 5 | 7 | 10 = | |
| Общая сумма | → | | | | |

**FP = общая сумма x [0,65 + 0,01 x SUM(Fj)]**

*Рис.24. Шаг 2 вычисления функциональных баллов*

Метрика в функциональных баллах, равно как и число строк программных кодов (LOC), в определенной степени противоречива. Убежденные противники функциональных баллов утверждают, что в этом случае имеет место некоторая подмена понятий, поскольку рассматриваемые вычисления основаны скорее на субъективных, нежели на объективных данных. Они же утверждают и то, что эти данные с трудом согласуются с фактическими показателями, полученными по завершении проекта. Кроме того, функциональные баллы не имеют физического содержания, каким обладает LOC. Сторонники этого метода приводят аргумент в их пользу, что функциональные баллы обладают преимуществом независимости от языков программирования. Это делает их более ценными по сравнению с числом LOC в отношении языков программирования четвертого поколения, которые являются непроцедурными и не связаны с понятием строки кода. Функциональные баллы можно получить непосредственно из технических требований, в то время как число LOC может быть получено на момент определения концепций только по аналогии с ранее разработанными проектами.

В настоящее время существует группа IFPUG (International Function Point Users Group— Международная группа пользователей функциональных баллов), которая была создана около десяти лет тому назад с целью стандартизации определений и использования рассматриваемой технологии. Есть и другие организации, занимающиеся разработкой программного обеспечения, которые применяют технологию функциональных баллов. В подавляющем большинстве проектов, которые выполняла наша организация, размеры программных продуктов оценивались числом строк программного кода, хотя для оценки размеров двух проектов были успешно применены функциональные баллы. При помощи таблицы перевода Кейперса Джонса (Capers Jones), представленной на рис. 12.1, можно перевести функциональные баллы в числа LOC для широкого спектра применяемых в настоящее время языков программирования.

# Резюме

В процессе планирования работ по выполнению проекта программного продукта необходимо решить следующие задачи:

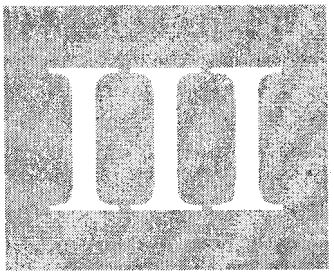
- Выбрать жизненный цикл разработки программного обеспечения.
- Достичь глубокого понимания предварительных требований и основных ограничений.
- Спрогнозировать трудозатраты и календарный график работ, а также дать оценку размеров и затрат средств на разработку, взяв за основу статистические данные по существующим программным продуктам.
- Воспользоваться подходом "разделяй и властвуй" с целью распределения задач среди предполагаемых исполнителей в соответствии с их квалификацией.

Соответствующий процесс получения оценки предусматривает следующие действия:

- Сбор входной информации, в том числе построение матрицы оценки размера программного обеспечения, и принятие решений относительно жизненного цикла разработки.

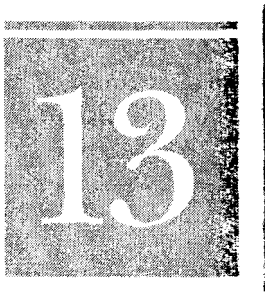
- Получение оценки общих трудозатрат (в часах) на основе статистических данных по существующим программным продуктам.
- Распространение оценки общих трудозатрат на каждую задачу и каждый рабочий график, соблюдая при этом границы основных стадий. Если возможно, оценку трудозатрат необходимо распространить и на различные сборки спиральной модели жизненного цикла.
- Документирование в форме электронной таблицы каждой задачи, трудозатрат на ее выполнение, графиков работ, включая информацию о базисе оценок и списки реальных условий, в которых реализуется жизненный цикл.
- Пересмотр оценок для передачи их руководству и выполнение предложенных корректив в соответствии с действующими требованиями.

ЧАСТЬ



# Примеры выполнения быстрого тестирования

# Пример формулирования требований




Глава 2 была посвящена процессу определения требований. В ней же приводилась диаграмма процесса формулирования требований, которая для удобства показывается еще раз на рис. 13.1.

В документе определения требований все системные требования излагаются на обычном разговорном языке. В результате упрощается работа с документом со стороны заказчиков, что позволяет их привлекать к процессу разработки продукта. Входными данными для документа определения требований является набор требований, сформированный в течения взятия неформальных интервью у заказчиков. Кроме того, проводятся более строгие совещания и целенаправленные сеансы, посвященные совместной разработке требований к приложению (JAR); организация JAR-сеансов подробно рассматривалась в главе 8.

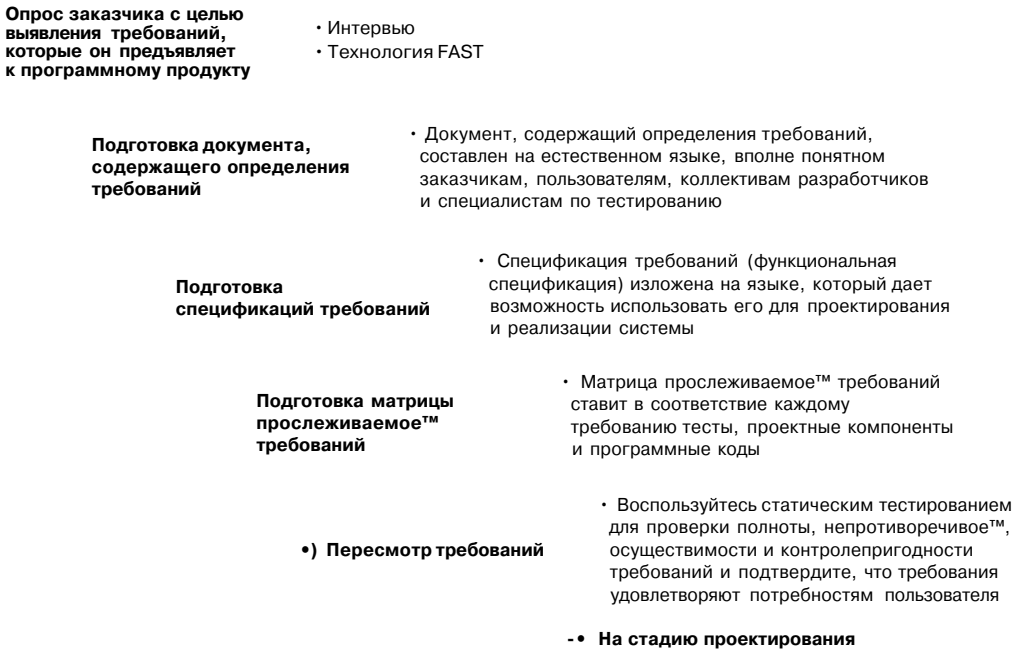



***Рис. 13.1. Процесс формулирования требований***

В главе 2 упоминалось о том, что определениям требований присущи следующие характеристики:

- Каждое требование должно быть снабжено уникальным идентификатором, чтобы можно было однозначно ссылаться на него при планировании тестового покрытия, при проектировании тестовых случаев и в отчетах по результатам тестирования.
- Требования должны быть представлены с точки зрения пользователя системы. Системные и приемочные испытания должны быть.спроектированы на основе определений требований, следовательно, эти определения должны быть сформулированы в перспективе системного уровня. Этот принцип препятствует появлению требований, затрагивающих внутренние свойства системы и требующих детальных знаний программного кода для успешного тестирования. Такие требования должны возникать на поздних стадиях разработки и охватываться модульным тестированием и проверкой взаимодействия и функционирования компонентов системы.
- Должны быть включены как *функциональные (functional),* так и *нефункциональные (nonfunctional)* требования. Функциональные требования суть требования, описывающие услуги и функции, которые должна выполнять разрабатываемая система. Нефункциональные требования описывают ограничения, накладываемые на работу системы, например, количество одновременно работающих пользователей, и стандарты, которым должна соответствовать система.
- Документ определения требований должен находится под управлением конфигурациями. Это, по меньшей мере, означает, что данный документ подпадает под управление версиями и что все версии документа должны быть помещены в безопасное хранилище, подобное, например, каталогу, содержимое которого обычно дублируется. Если требования подвергаются изменениям, мы должны иметь возможность проследить, чтобы соответствующие изменения были внесены в тестовые случаи системных и приемочных испытаний.

Структура документа определения требований может основываться на спецификациях, определенных в стандарте *IEEE Standard 830: The IEEE Guide to Software Requirements Specifications* (дополнительные сведения по применению этого стандарта можно найти в главе 2).

Сейчас мы предлагаем приступим к изучить пример документа определения требований, который был подготовлен для вымышленного набора инструментальных средств управления тестированием (Test Management Toolkit, TMT). Это приложение позволяет менеджерам и инженерам, специалистам в области тестирования управлять планами тестирования, отчетами о дефектах, результатами тестирования, а также другой информацией, связанной с тестированием программного обеспечения. TMT представляет собой Web-приложение, благодаря чему несколько географически удаленных пользователей могут одновременно участвовать в нескольких проектах по тестированию. Проект разработки приложения TMT приводится в качестве примера, который рассматривается на протяжении всей третьей части книги. В следующей главе содержится пример плана тестирования, который основан на требованиях к приложению TMT, определенных в данной главе.

**Идентификатор документа:** TMT-RD-10
**Версия: 0.8**
**Автор:** Крис Браун (Chris Brown)

### Набор инструментальных средств управления тестированием

### Версия 1.0

# Определение требований

## Хронология версий

| Версия | Дата | Автор | Описание |
|---|---|---|---|
| 0.1 | 08/31/2001 | Крис Браун | Эскиз документа определения требований для TMT |
| 0.2 | 09/02/2001 | Крис Браун | Добавлена возможность управления проектами, выполнена доводка |
| 0.3 | 09/02/2001 | Крис Браун | Расширена структура данных, выполнена доводка |
| 0.4 | 09/03/2001 | Крис Браун | Добавлена возможность создания матрицы прослеживаемости требований |
| 0.5 | 09/03/2001 | Крис Браун | Добавлены экраны справки |
| 0.6 | 09/03/2001 | Крис Браун | В структуру данных добавлено поле Project Name (Название проекта) |
| 0.7 | 09/07/2001 | Крис Браун | Добавлено требование многопользовательской поддержки |
| 0.8 | 09/07/2001 | Крис Браун | Добавлена многопользовательская поддержка в индексный раздел 3 |

## Утверждено

| | | Дата утверждения |
|---|---|---|
| Маркетинга | Чак Д. Клут (Chuck D. Klout), директор отдела маркетинга | 09/07/2001 |
| Программирования | Сюзанна Перл (Suzanna Perl), менеджер отдела программирования | 09/09/2001 |
| Тестирования | Брит Гейтер (Bret Gater), менеджер отдела тестирования | 09/09/2001 |

# Содержание

## 1. Введение

### *1.1. Назначение*

Целью создания этого документа является определение набора требований к программному продукту, именуемому набором инструментальных средств управления тестированием (Test Management Toolkit, TMT). Документ предназначен для сотрудников отделов маркетинга, программирования, тестирования, а также для технического персонала, осуществляющего поддержку программных продуктов. Документ организован таким образом, что обеспечивает возможность выделять, идентифицировать и выбирать отдельные требования. Требования излагаются на таком уровне детализации, что на их основе разработчики могут создавать программный продукт, а тестировщики — выполнять аттестацию этого продукта. Программный продукт TMT может использоваться для упрощения процессов тестирования оборудования, программного обеспечения и системы в целом. С целью упрощения все примеры применения TMT ограничиваются тестированием программного обеспечения. Этот документ предназначен только для внутреннего использования.

### *1.2. Область применения*

Приложение TMT обеспечивает для менеджеров и инженеров по тестированию возможность управления планами тестирования, отчетами о дефектах, результатами прогона тестов и другой информацией, связанной с тестированием программного обеспечения. TMT представляет собой Web-приложение, благодаря чему несколько географически удаленных пользователей могут одновременно участвовать в нескольких проектах по тестированию.

### *1.3. Обзор*

Первые разделы документа содержат общее описание приложения TMT, а в следующих разделах приводится список детализованных требований к этому программному продукту. Каждое требование снабжено уникальным идентификатором, благодаря чему разработчики могут прослеживать трудозатраты на разработку и тестирование обратно в направлении требований. Дескрипторы требований представлены в форме заголовков разделов. Например, раздел 2.2.1 представляет требование, которое определяет методику применения приложения TMT для обработки документов, связанных с планом тестирования. На дескриптор "2.2.Г можно ссылаться в других проектных документах. В результате появляется гарантия, что более поздние ссылки на это требование могут отслеживаться в обратном направлении вплоть до исходного требования, которое содержится в данном документе.

## 2. Общее описание

### *2.1. Перспективы применения продукта*

Приложение TMT предназначено для упрощения тестирования программного обеспечения. Это приложение может оказать существенную помощь при разработке, выполнении и отслеживании тестируемых программ. Этот документ описания требований направляет непосредственно на свойства, которыми должен обладать коммерческий продукт, ориентированный на применение в области разработки и тестирования программного обеспечения.

### *2.2. Функции продукта*

В этом разделе описываются высокоуровневые функциональные свойства программного продукта TMT. Более подробное описание требований находится в разделе 3.0.

#### 2.2.1. **Тестовые планы**

Продукт TMT должен обеспечивать средства создания, изменения, просмотра, сохранения и выборки документов, описывающих тестовые планы.

#### **2.2.2. Список прогона**

Продукт TMT должен поддерживать средства создания, изменения, просмотра, сохранения и выборки набора тестов для прогона. В рамках данного документа этот набор тестов именуется "списком прогона".

#### 2.2.3. Тесты

Продукт TMT должен обеспечивать средства создания, изменения, просмотра, сохранения и выборки отдельных тестов, которые могут включать такие компоненты, как процедуры установки, списки оборудования, процедуры тестирования, тестовые данные, а также процедуры, обеспечивающие окончательную доводку (очистку) данных и результатов прогона тестов.

#### **2.2.4. Выполнение тестов**

Продукт TMT должен поддерживать средства запуска на выполнение тестов из списка прогона. При этом для каждого теста могут создаваться свои результаты и журнальные файлы.

#### 2.2.5. Отчеты об ошибках

Продукт TMT должен поддерживать средства создания, изменения, просмотра, сохранения и выборки отчетов об ошибках.

#### 2.2.6. Итоговые отчеты

Продукт TMT должен обеспечивать средства построения отчетов, которые подводят итоги по состоянию тестирования, результатам прогона тестов и отчетам об обнаруженных дефектах.

#### 2.2.7. Резервное копирование и восстановление

Продукт TMT должен иметь средства, предназначенные для резервного копирования и восстановления как тестов, так и результатов тестирования.

#### 2.2.8. Безопасность

Продукт TMT должен поддерживать средства создания учетных записей и паролей для пользователей приложения. Для запуска тестов на выполнение и просмотра результатов прогона требуется аутентификация пользователей.

#### 2.2.9. Удаленное администрирование

В составе программного продукта TMT должен присутствовать модуль, обеспечивающий удаленное администрирование. Менеджер, руководитель и системный администратор должны располагать возможностями реализации административных функций из удаленных рабочих мест в отношении базы данных и пользовательской информации.

#### 2.2.10. Матрица прослеживаемости требований

В состав программного продукта TMT должен быть включен модуль, предназначенный для создания матрицы прослеживаемости требований. Он должен выглядеть как каркас или набор пустых заполнителей, которые после постепенного помещения действительных номеров и описаний всех требований превращается в результирующую матрицу прослеживаемости.

### *2.3. Пользовательские характеристики*

Пользователями описываемого программного продукта являются менеджеры, руководители, инженеры и технические работники, занимающиеся тестированием программного обеспечения.

### *2.4. Общие ограничения*

Ниже перечислены ограничения, могущие повлиять на возможности команды разработчиков программного обеспечения:

- Регулирующие политики.
- Ограничения, связанные с оборудованием.
- Интерфейсы с другими приложениями.
- Требования, накладываемые языками высокого уровня.
- Протоколы.
- Критический режим функционирования продукта.

### 2.5. *Допущения и зависимости*

В этом разделе описаны допущения и зависимости, связанные с программным продуктом TMT. С целью упрощения будущих ссылок каждое допущение или зависимость помечается специальным идентификатором (заголовком раздела). Перечисленные допущения следует принимать во внимание во время создания конфигураций тестируемых систем с использованием TMT.

#### 2.5.1. Операционные системы

Предполагается, что пользователь выполняет клиентское приложение на компьютере, работающем под управлением одной из следующих операционных систем:

Microsoft

- Windows 95
- Windows 98
- Windows Millennium Edition
- Windows XP
- Windows NT 3/51 или выше
- Windows 2000.

Apple

- MAC OS 9.x или выше.

Предполагается, что серверное приложение выполняется на компьютере, работающем под управлением одной из следующих операционных систем:

Microsoft

- Windows NT 3.51 или выше.

UNIX

- Sun Solaris 2.6 или выше
- HPUX 10.x или выше
- Open BSD
- AIX 2.4.1 или выше
- SCO Open Desktop
- Linux Red Hat 6.x или выше.

#### 2.5.2. Браузеры

Предполагается, что пользователь на клиентском компьютере использует один из следующих браузеров: Netscape версии 4.0 или выше, Internet Explorer версии 5.0 или выше.

#### 2.5.3. Реляционные базы данных

Предполагается, что на хост-компьютере (сервере) выполняется программное обеспечение реляционных баз данных, которое допускает использование множественной индексации. Также предполагается, что программное обеспечение баз данных обладает свойством блокирования записей. Кроме того, необходимо, чтобы это программное обеспечение включало в свой состав библиотеки API, совместимые со стандартным языком SQL. В начальной версии продукта TMT предполагается использование баз данных Oracle. В будущих версиях TMT могут применяться другие системы управления базами данных, что должно регламентироваться отзывами пользователей, касающимися первой версии продукта.

#### 2.5.4. Сервер Web-страниц

Предполагается, что на хост-компьютере (сервере) выполняется серверное приложение Web-страниц, например, Apache или Microsoft Internet Information Server.

#### 2.5.5. Зависимость от процессора

Приложение не зависит от типа применяемого процессора. Перечисленные ранее допустимые операционные системы могут использоваться на платформах с процессорами x86, RISC, SPARC, Motorola или PPC.

## 3. Специальные требования

В этом разделе представлены детализованные требования, относящиеся к программному продукту TMT.

### *3.1. Функциональные требования*

#### 3.1.1. Пользовательский интерфейс

Пользовательский интерфейс для клиента TMT создается с использованием языка HTML и отображается в окне Web-браузера. Применение HTML уменьшает степень зависимости от браузеров.

#### 3.1.2. Навигация

Главное меню программного продукта TMT включает следующие пункты:

**Current Projects (Текущие проекты)**
**Completed Projects (Завершенные проекты)**
**Project Maintenance (Сопровождение проекта)**
Create New Project (Создать новый проект)
*Modify Project (Изменить проект)*
*Remove Project (Удалить проект)*
*Help (Справка)*
**Test Case Maintenance (Сопровождение тестовых случаев)**
*Create Test Case or Suite (Создать тестовый случай или набор)*
*Modify Test Case or Suite (Изменить тестовый случай или набор)*
*Remove Test Case or Suite (Удалить тестовый случай или набор)*
*Display Test (Показать тест)*
*Display Suite (Показать тестовый набор)*
*Help (Справка)*
**Test Case Execution (Выполнение тестовых случаев)**
*Run Single Test (Прогнать одиночный тест)*
*Run Suite (Прогнать тестовый набор)*
*Create Run List (Создать список прогона)*
*Execute Run List (Выполнить список прогона)*
**Test Results (Результаты тестирования)**
*Bug Summary (Сводный отчет по ошибкам)*
*Single Test (Одиночный тест)*
*Suite or Run List (Тестовый набор или список прогона)*
*Help (Справка)*
**Utilities (Утилиты)**
*Create Trace Matrix (Создать матрицу прослеживаемости)*
**Backup (Резервное копирование)**
Tesf *Cases (Тестовые случаи)*
*Test Suites (Тестовые наборы)*
*Test Results (Результаты прогона тестов)*
*Help (Справка)*
**Restore (Восстановление)**
Tesf *Cases (Тестовые случаи)*
*Test Suites (Тестовые наборы)*
*Test Results (Результаты прогона тестов)*
*Help (Справка)*
**Export (Экспорт)**
Tesf Cases *(Тестовые случаи)*
*Test Suites (Тестовые наборы)*
*Test Results (Результаты прогона тестов)*
*Help (Справка)*

### 3.1.3. Аутентификация пользователей — клиент

Клиентский сеанс открывается после регистрации на хост-системе через Internet или интрасеть компании. Для прогона тестов и просмотра результатов пользователи должны вводить свои имена и пароли. Имя пользователя и пароль должны быть уникальными и присваиваться администратором. Имя и пароль не могут изменяться самим пользователем.

### 3.1.4. Аутентификация пользователей — администратор

Менеджер, руководитель и администратор должны вводить имя пользователя и пароль для того, чтобы получить доступ к пользовательской информации либо выполнить администрирование базы данных. Имя пользователя и пароль должны быть уникальными и могут изменяться только администратором.

### 3.1.5. Текущие проекты

После регистрации в системе пользователь получает возможность просматривать активные и текущие проекты. В результате выбора этой опции из главного меню отображается список активных проектов. В названии проекта отображается связанное с ним общее количество тестов либо тестовых наборов. После общего числа тестов/тестовых наборов указывается процент выполненных тестов/тестовых наборов. Затем выводится количество тестов, которые прошли. Далее выводится процентное отношение количества пройденных тестов к общему их числу. Затем отображается количество тестов, завершившихся неудачно. После информации о количестве тестов, завершившихся неудачно, выводится их процентное отношение к общему числу тестов. Далее отображается общее число заблокированных тестов. После общего числа заблокированных тестов указывается их процентное отношение к общему числу тестов. Затем выводится время, оставшееся до завершения данного проекта.

Время, которое осталось до завершения проекта, может описываться как "Not Known" ("Неизвестное") или же указываться в часах и минутах. Время выполнения теста определяется временем его последнего прогона. Оставшийся запас времени представляет собой общее значение времени (сумма по всем тестам) минус время, затраченное на уже выполненные тесты.

### 3.1.6. Завершенные проекты

После регистрации в системе пользователь получает возможность просматривать завершенные проекты. В результате выбора этой опции из главного меню отображается список завершенных проектов. После названия проекта указывается общее количество тестов или тестовых наборов, связанных с проектом. Затем, после общего числа тестов/тестовых наборов отображается процент выполнения данного проекта. Следом за процентом выполнения указывается количество успешно пройденных тестов. После количества пройденных тестов отображается их процент в общем количестве тестов. За значением процента следует число тестов, потерпевших неудачу. После этого отображается их процентное отношение к общему количеству тестов, а затем — общее число заблокированных тестов. После общего количества заблокированных тестов указывается их процентное отношение к общему числу тестов. Наконец, отображается время прогона всех тестов для данного проекта.

### 3.1.7. Создание нового проекта

После регистрации в системе пользователь получает возможность создавать новый проект. Процесс создания проекта начинается непосредственно после выбора пользователем опции "Create New Project" ("Создать новый проект") из меню "Project Maintenance" ("Сопровождение проекта"). Вначале предлагается ввести имя нового проекта. После ввода имени проекта отображается список всех доступных тестовых случаев и наборов.

После выбора тестовых случаев и/или тестовых наборов, пользователь должен получить возможность сохранить проект или отменить действия и вернуться в главное меню. При выборе сохранения список тестовых случаев и/или тестовых наборов должен быть сохранен как проект.

#### 3.1.8. Изменение проекта

После регистрации в системе пользователь должен иметь возможность изменять существующий проект. Процесс изменения начинается после выбора элемента "Modify Project" ("Изменить проект") из меню "Project Maintenance" ("Сопровождение проекта"). На этом этапе отображается запрос на ввод имени проекта, а также предлагается список доступных проектов, среди которых можно произвести выбор. Пользователь должен либо ввести имя проекта, либо дважды щелкнуть на соответствующем имени проекта в списке.

В случае отсутствия проектов отображается сообщение "No Projects Have Been Created" ("Созданные проекты отсутствуют"). На этом этапе пользователь должен иметь только один вариант выбора, которым служит кнопка "OK", щелчок на которой закрывает окно сообщения и приводит к возврату в главное меню.

#### 3.1.9. Удаление проекта

Процесс удаления проекта начинается после выбора пользователем элемента "Remove Project" ("Удалить проект") из меню "Project Maintenance" ("Сопровождение проекта"). Пользователю выдается запрос об имени удаляемого проекта, а также список доступных проектов, в котором можно произвести выбор. Пользователь должен иметь возможность либо ввести имя удаляемого проекта, либо дважды щелкнуть на имени соответствующего проекта в списке.

В случае отсутствия проектов отображается сообщение "No Projects Have Been Created" ("Созданные проекты отсутствуют"). На этом этапе пользователь должен иметь только один вариант выбора, которым служит кнопка "OK", щелчок на которой закрывает окно сообщения и приводит к возврату в главное меню.

#### 3.1.10. Создание тестового случая или набора

После регистрации в системе пользователь должен иметь возможность создавать новые тестовые случаи или наборы. Процесс создания начинается после выбора пользователем опции "Create Test Case or Suite Test" ("Создать тестовый случай или набор ") из меню "Test Case Maintenance" ("Сопровождение тестовых случаев").

Если проект уже идентифицирован, пользователь должен получить запрос на ввод имени проекта либо выбрать его из списка. Если до этого момента не было идентифицировано ни одного проекта или тестового набора, пользователь должен указать, является ли его целью создание одиночного тестового случая или части тестового набора. При выборе одиночного тестового случая отображается запрос на ввод его имени. Если выбран тестовый набор, потребуется ввести имя этого набора.

**Тестовый случай:**

Для пользователя выводится интерфейс в виде формы с предварительно определенными полями, в которые потребуется ввести необходимые данные. На этом этапе форма именуется "корзиной", и она является представлением создаваемой структуры типа записи. После ввода данных в запись, пользователь должен иметь возможность сохранить тест или отменить действие. В случае выбора сохранения, запись данных помещается в базу данных. Кроме того, пользователь должен иметь возможность распечатать тестовый случай.

Тестовый набор:

Если пользователь выбирает тестовый набор, отображается запрос на ввод имени набора. Затем отображается список доступных (уже созданных) тестовых случаев. Достаточно будет выбрать тестовые случаи, установив соответствующие флажки. После выбора пользователь должен иметь возможность сохранить тестовый набор или отменить действие. Если выбрано сохранение, создается тестовый набор, который помещается в базу данных. Кроме того, пользователь должен иметь возможность распечатать тестовый набор.

#### 3.1.11. Изменение тестового случая или набора

После регистрации в системе пользователь должен иметь возможность изменять существующие тестовые случаи или наборы. Процесс изменения начинается после выбора опции "Modify Test Case

or Suite" ("Изменить тестовый случай или набор ") в меню "Test Case Maintenance" ("Сопровождение тестовых случаев").

Далее пользователь должен ввести имя теста или тестового набора. При этом должна быть возможность выбора соответствующего имени из списка. В случае отсутствия тестов или тестовых наборов отображается сообщение "There are no Test Cases or Suites to Modify" ("Не существует тестовых случаев или наборов для изменения").

Тестовый случай:

Пользователь должен получить доступ к тесту, который находится в том же формате, что и в момент создания. При этом поддерживается интерфейс в виде формы с предварительно определенными полями, содержащими предварительно определенные данные. Если тестовый случай изменялся, пользователь должен иметь возможность сохранить измененный тест или отменить действие. При выборе операции сохранения запись помещается в базу данных. Кроме того, должна существовать возможность распечатки тестового случая.

**Тестовый набор:**

Если пользователь выбирает тестовый набор для изменения, отображаются все доступные тесты, в число которых входят и уже выбранные. Последние идентифицируются отмеченными флажками.

Пользователь может выбрать дополнительные тесты либо исключить ранее выбранные. По завершении процесса выбора должны быть доступны три возможности. Измененный тестовый набор можно сохранить под тем же именем либо определить для него новое имя. После выбора операции сохранения тестовый набор закрывается, а изменения помещаются в базу данных. Кроме того, пользователь должен иметь возможность прервать действие и отменить все проделанные изменения. В дополнение должна существовать возможность распечатки тестового набора.

### 3.1.12. Удаление тестового случая или набора

Процесс удаления инициируется после выбора пользователем опции "Remove Test Case or Suite" ("Удалить тестовый случай или набор ") из меню "Test Case Maintenance" ("Сопровождение тестовых случаев"). В результате появляется возможность удалить отдельный тестовый случай или целый набор. Выбор производится в отображаемом списке тестовых наборов и тестовых случаев, при этом с каждым элементом связан отдельный флажок. Пользователю потребуется просто отметить те тесты, которые должны быть удалены.

Далее у пользователя есть возможность удалить выделенные тесты либо отменить действие. После выбора операции удаления отображается дополнительный запрос "Are you sure?" ("Вы уверены?"). Пользователь может выбрать опцию "Yes, Delete the selected entries" ("Да, удалить выделенные элементы") или отменить действие. В случае выбора удаления происходит действительное удаление соответствующих записей. Список удаляемых файлов можно вывести на печать.

### 3.1.13. Отображение теста

Эта опция обеспечивает возможность просмотра детальной информации, связанной с тестовым случаем. После выбора пользователем из меню "Test Case Maintenance" ("Сопровождение тестовых случаев") опции "Display Test" ("Показать тест") отображается список доступных тестов. Для просмотра теста достаточно выполнить двойной щелчок кнопкой мыши на его имени. Тестовый случай будет отображаться в том формате, в котором он пребывал на этапе создания. Выводимые данные можно только просматривать, но не изменять. Кроме того, тестовый случай можно распечатать.

### 3.1.14. Отображение тестового набора

Эта опция обеспечивает возможность просмотра тестового набора. После выбора пользователем элемента "Display Suite" ("Показать тестовый набор") из меню "Test Case Maintenance" ("Сопровождение тестовых случаев") отображается список доступных тестовых наборов. Для просмотра требуемого тестового набора необходимо дважды щелкнуть кнопкой мыши на его имени. После этого тестовый набор выводится в форме списка тестовых случаев, при этом будут отображаться только те тестовые случаи, для которых были отмечены флажки во время создания. Данная опция обеспе-

чивает только просмотр, поэтому внесение каких-либо изменений в состав тестового набора невозможно. Тестовый набор можно вывести на печать.

#### 3.1.15. Прогон одиночного теста

Эта опция активизируется после выбора пользователем элемента "Run Single Test" ("Прогнать одиночный тест") из меню "Test Case Maintenance" ("Сопровождение тестовых случаев"). Для пользователя должен выводиться список доступных тестов. Требуемый тестовый случай выбирается двойным щелчком кнопкой мыши. Прогонять можно как автоматизированные, так и тесты, выполняемые вручную.

**Тестирование, выполняемое вручную:**

Тестовый случай отображается в режиме только для чтения со множеством опций в нижней части экрана. Предполагается, что специалист по тестированию выполнит действия, детально описанные в тестовом случае, и посмотрит на полученные результаты. В частности, доступны опции "Pass" ("Прошел"), "Fail" ("Не прошел") либо "Blocked" ("Заблокирован"). После выбора опции "Fail" или "Blocked" выводится запрос о причинах, который содержит текстовое поле для описания причин неуспешного прогона тестового случая. Совершенно аналогично вводятся детали для тестового случая с признаком "Blocked". После описания причин неудачного прогона теста пользователь должен выбрать опцию "Finish" ("Готово"), в результате чего результат регистрируется.

**Автоматическое тестирование:**

Если тест является автоматизированным, после двойного щелчка кнопкой мыши запускается сценарий, реализованный на Perl, TCL или другом языке написания сценариев. Тестовый случай выполняется, а результаты регистрируются автоматически.

#### 3.1.16. Прогон тестового набора

Эта опция активизируется после выбора из меню "Test Case Execution" ("Выполнение тестовых случаев") элемента "Run Suite" ("Прогнать тестовый набор"). Для пользователя отображается список доступных тестовых наборов. Для запуска на выполнение требуемого тестового набора тестов достаточно выполнить двойной щелчок кнопкой мыши на его имени. В результате будет иметь место одно из двух возможных действий. Тестовые наборы могут быть предназначены для прогона в ручном режиме и отображаться по очереди. Пользователю предоставляется возможность выполнить все шаги вручную и указать результат в форм "прошел/не прошел". Затем отображается следующий тест и все повторяется сначала.

Если тестовые наборы являются автоматизированными, все тестовые случаи прогоняются, а результаты регистрируются.

Кроме того, возможны комбинации автоматизированных тестов и тестами, прогоняемых вручную. Все зависит от деталей отдельных тестовых случаев, выбираемых во время создания тестового набора.

#### 3.1.17. Создание списка прогона

Данная опция дает возможность пользователю генерировать список тестов, которые можно выбирать среди тестовых случаев, содержащихся в любых тестовых наборах. Каждый список прогона получает имя и сохраняется для будущего выполнения.

#### 3.1.18. Выполнение списка прогона

Данная опция дает возможность пользователю выбрать и запустить на выполнение любой ранее сохраненный список прогона.

#### 3.1.19. Сводный отчет по ошибкам

В результате выбора этой опции пользователь получает запрос относительно выбора текущего или завершенного проекта для его анализа. Опция находится в меню Tests Results (Результаты тестирования). В случае выбора из текущих проектов, отображается список текущих проектов, а в случае

выбора из завершенных проектов - соответственно, список завершенных проектов. Для анализа какого-либо проекта следует дважды щелкнуть кнопкой мыши на его имени. При этом отображается подробная информация о найденных ошибках, организованная по их степени серьезности. Отчет такого рода позволяет сделать вывод относительно уровня подготовленности конкретного проекта, а также стабильности его дальнейшего функционирования.

#### 3.1.20. Результаты тестирования — одиночный тест

Эта опция приводит к выводу списка результатов прогона тестового случая. Опция находится в меню Tests Results (Результаты тестирования). Отображаемая информация может сортироваться по идентификатору теста, дате тестирования, времени тестирования и состоянию теста, которым может "pass" ("прошел"), "fail" ("не прошел") и "blocked" ("заблокирован"). Допускается выделять диапазоны тестов и генерировать итоги.

#### 3.1.21. Результаты тестирования — тестовый набор или список прогона

Эта опция дает возможность получить список тестовых наборов или списков прогона, которые выполнялись или выполняются в настоящий момент. Опция доступна через меню Tests Results (Результаты тестирования). В результате отображаются итоги по тестам, которые завершились удачно, неудачно и были заблокированы, а также процент завершения тестирования.

#### 3.1.22. Создание матрицы прослеживаемости

Эта опция предоставляет возможность генерации матрицы прослеживаемости требований (Requirements Traceability Matrix), которая используется на этапах планирования и выполнения тестов. Активизация опции выполняется после выбора пользователем элемента "Create Trace Matrix" ("Создать матрицу прослеживаемости") из меню "Utilities" ("Утилиты"). Пользователю выдается запрос на ввод имени проекта. При этом можно просто ввести имя проекта либо дважды щелкнуть на требуемом имени в отображаемом списке доступных проектов. Создается экранный отчет с уже определенным списком требований и местами для помещения информации, связанной с тестовыми случаями. На этом экране также имеется возможность отправить матрицу на принтер или сохранить ее в формате электронных таблиц, совместимых с Excel.

Если еще не было создано ни одного проекта, отображается сообщение "No Projects Have Been Created" ("Созданные проекты отсутствуют"). На этом этапе пользователю доступна только одна возможность - выполнить щелчок на кнопке "OK" и после чего вернуться в главное меню.

#### 3.1.23. Резервное копирование тестовых случаев

Эта опция дает возможность пользователю создавать архивные резервные копии тестовых случаев. Опция активизируется в результате выбора элемента "Backup/Test Cases" ("Резервное копирование/Тестовые случаи") из меню "Utilities" ("Утилиты"). Пользователь получает запрос на ввод имени резервной копии и целевого носителя. Эти данные отличаются от системы к системе. В среде Windows отображается стандартный список физических, логических и подключенных сетевых дисков. В системе UNIX можно получить список псевдонимов смонтированных томов, в рамках которого и произвести свой выбор. Учитывая размеры файлов резервного копирования, обыкновенные дискеты редко когда подходят. В системах, в состав которых входят дисководы CD/R или CD/RW, можно просто выполнить резервное копирование на такое устройство, не используя при этом специализированного программного обеспечения для записи CD/RW.

#### 3.1.24. Резервное копирование тестовых наборов

Данная опция предоставляет пользователю возможность создавать архивные резервные копии тестовых наборов. Опция активизируется после выбора пользователем элемента "Backup/Test Suites" ("Резервное копирование/Тестовые наборы") из меню "Utilities" ("Утилиты"). Пользователь получает запрос на ввод имени резервной копии и целевого носителя. Эти данные отличаются от системы к системе. В среде Windows отображается стандартный список физических, логических и подключенных сетевых дисков. В системе UNIX можно получить список псевдонимов смонтированных томов, в

рамках которого и произвести свой выбор. Учитывая размеры файлов резервного копирования, обыкновенные дискеты редко когда подходят. В системах, в состав которых входят дисководы CD/R или CD/RW, можно просто выполнить резервное копирование на такое устройство, не используя при этом специализированного программного обеспечения для записи CD/RW.

#### 3.1.25. Резервное копирование результатов прогона тестов

Эта опция предоставляет пользователю возможность создавать архивные резервные копии результатов прогона тестов. Опция активизируется после выбора пользователем элемента "Backup/Test Results" ("Резервное копирование/Результаты прогона тестов") из меню "Utilities" ("Утилиты"). Пользователь получает запрос на ввод имени резервной копии и целевого носителя. Эти данные отличаются от системы к системе. В среде Windows отображается стандартный список физических, логических и подключенных сетевых дисков. В системе UNIX можно получить список псевдонимов смонтированных томов, в рамках которого и произвести свой выбор. Учитывая размеры файлов резервного копирования, обыкновенные дискеты редко когда подходят. В системах, в состав которых входят дисководы CD/R или CD/RW, можно просто выполнить резервное копирование на такое устройство, не используя при этом специализированного программного обеспечения для записи CD/RW.

#### 3.1.26. Восстановление тестовых случаев

Эта опция дает возможность восстанавливать тестовые случаи, для которых ранее создавались резервные копии. Опция активизируется после выбора элемента "Restore/Test Cases" ("Восстановление/Тестовые случаи") из меню "Utilities" ("Утилиты"). Пользователь должен выбрать источник расположения и имя восстанавливаемой резервной копии.

#### 3.1.27. Восстановление тестовых наборов

Эта опция дает возможность восстанавливать тестовые наборы, для которых ранее создавались резервные копии. Опция активизируется после выбора элемента "Restore/Test Suites" ("Восстановление/Тестовые наборы ") из меню "Utilities" ("Утилиты"). Пользователь должен выбрать источник расположения и имя восстанавливаемой резервной копии.

#### 3.1.28. Восстановление результатов прогона тестов

Эта опция дает возможность восстанавливать результаты прогона тестов, для которых ранее создавались резервные копии. Опция активизируется после выбора элемента "Restore/Test Results" ("Восстановление/Результаты прогона тестов") из меню "Utilities" ("Утилиты"). Пользователь должен выбрать источник расположения и имя восстанавливаемой резервной копии.

#### 3.1.29. Экспорт тестовых случаев

Эта опция предоставляет возможность экспортировать выбранные тестовые случаи в файлы ASCII-формата с разделителями-запятыми. Опция активизируется после выбора элемента "Export/Test Cases" ("Экспорт/Тестовые случаи") из меню "Utilities" ("Утилиты"). В результате отображается список всех доступных тестовых случаев. Для выполнения экспорта пользователю достаточно просто дважды щелкнуть на имени требуемого тестового случая. Далее выдается запрос на указание целевого местоположения и имени выходного файла. Пользователь должен иметь возможность отправлять данные экспорта на принтер.

#### 3.1.30. Экспорт тестовых наборов

Эта опция предоставляет возможность экспортировать выбранные тестовые наборы, разделенные на отдельные тестовые наборы, в файлы ASCII-формата с разделителями-запятыми. Опция активизируется после выбора элемента "Export/Test Suites" ("Экспорт/Тестовые наборы") из меню "Utilities" ("Утилиты"). В результате отображается список всех доступных тестовых наборов. Для выполнения экспорта пользователю достаточно просто дважды щелкнуть на имени требуемого тестового набора. Далее выдается запрос на указание целевого местоположения и имени выходного файла. Пользователь должен иметь возможность отправлять данные экспорта на принтер.

#### 3.1.31. Экспорт результатов прогона тестов

Эта опция предоставляет возможность экспортировать выбранные результаты прогона тестов в файлы ASCII-формата с разделителями-запятыми. Опция активизируется после выбора элемента "Export/Test Results" ("Экспорт/Результаты прогона тестов") из меню "Utilities" ("Утилиты"). После этого отображается список всех доступных результатов прогона тестов. Для выполнения экспорта пользователю достаточно просто дважды щелкнуть на имени требуемого результата прогона тестов. Далее выдается запрос на указание целевого местоположения и имени выходного файла. Пользователь должен иметь возможность отправлять данные экспорта на принтер.

#### 3.1.32. Справка

Для каждого элемента меню должен существовать экран с соответствующей справочной информацией по возможностям, реализуемым тем или иным элементом меню. Содержимое справочного экрана всецело зависит от меню, из которого он вызывается.

#### 3.1.33. Многопользовательские функциональные возможности

Данный программный продукт должен функционировать в коммерческой, сетевой среде и обеспечивать множеству пользователей возможность одновременного тестирования нескольких проектов. Следует убедиться, что продукт поддерживает приемлемые показатели производительности при работе с ним нескольких тестировщиков. Проведенные маркетинговые исследования дают возможность заключить, что одновременная работа до пяти тестировщиков должна будет приветствоваться сообществом пользователей.

### *3.2. Требования к внешнему интерфейсу*

Какие-либо требования к аппаратным или программным внешним интерфейсам отсутствуют.

### *3.3. Требований к производительности*

В этом разделе описывается требования к производительности программного продукта TMT.

#### 3.3.1. Возможность поддержки нескольких пользователей

Продукт TMT предназначен для функционирования в многопользовательском режиме. Поскольку продукт разрабатывается впервые и ориентируется на внутреннее применение, он не рассчитан на перегрузки, характерные для коммерческих Web-приложений. Тем не менее, важно удостовериться в возможности поддержки пяти клиентских сеансов без существенного снижения производительности. Во время тестирования должна оцениваться производительность для случаев работы от одного до пяти пользователей, которые одновременно выполняют как аналогичные, так и различные виды работ.

### *3.4. Ограничения проекта*

В начальной версии продукта TMT используется приложение базы данных Oracle. Проектирование и реализация программного продукта должны быть совместимы с Oracle.

### 3.5. *Структура данных*

Проект структуры данных для тестового случая показан в таблице 3.1, а активные индексы — в таблице 3.2.

### *3.6. Атрибуты*

Какие-либо требования к сопровождению и переносимости отсутствуют.

### *3.7. Прочие требования*

Прочие требования в настоящий момент отсутствуют.

**Таблица 3.1. Структура данных для представления тестового случая**

| Описание поля | Метка поля | Тип поля | Размер поля |
|---|---|---|---|
| Project Name (Имя проекта) | PRJNAM | Character (Символьный) | 20 |
| Requirement Satisfied (Удовлетворенное требование) | REQSAT | Numeric (Числовой) | 8 |
| Requirement Name (Имя требования) | REQNAM | Character | 20 |
| Test Identifier (Идентификатор теста) | TSTID | Character | 20 |
| Test Name (Имя теста) | TSTNAM | Character | 20 |
| Expected Result (Ожидаемый результат) | XRSLT | Character | 50 |
| Hardware Required (Требуемое оборудование) | HWREQ | Character | 256 |
| Test Setup (Настройка теста) | TSTSUP | Character | 256 |
| Configuration Name (Имя конфигурации) | CFGNUM | Character | 20 |
| Test 1 Name (Имя теста 1) | TST1NUM | Character | 20 |
| Test 1 Steps (Шаги теста 1) | TST1STP | Character | 256 |
| Test 2 Name (Имя теста 2) | TST2NUM | Character | 20 |
| Test 2 Steps (Шаги теста 2) | TST2STP | Character | 256 |
| Test 3 Name (Имя теста 3) | TST3NUM | Character | 20 |
| Test 3 Steps (Шаги теста 3) | TST3STP | Character | 256 |
| Test 4 Name (Имя теста 4) | TST4NUM | Character | 20 |
| Test 4 Steps (Шаги теста 4) | TST4STP | Character | 256 |
| Test 5 Name (Имя теста 5) | TST5NUM | Character | 20 |
| Test 5 Steps (Шаги теста 5) | TST5STP | Character | 256 |
| User ID (Идентификатор пользователя) | UID | Character | 20 |
| Date (Дата) | DATRUN | Date (Дата) | 8 |
| Time Start (Время начала) | START | Time (Время) | 4 |
| Time Stop (Время окончания) | STOP | Time | 4 |
| Time Needed (Требуемое время) | TIMREQ | Numeric | 4 |
| Pass(Пройден) | PASS | Logical (Логический) | T/F (истина/ ложь) |
| Fail (Отказ) | FAIL | Logical | T/F |
| Fail Detail (Детали отказа) | FLDTL | Character | 256 |
| Blocked (Заблокирован) | BLCKD | Logical | T/F |
| Blocked Detail (Детали блокировки) | BLKDTL | Character | 256 |
| Post Test Cleanup (Очистка по завершении теста) | CLNUP | Character | 50 |

**Таблица 3.2. Индексы в базе данных тестовых случаев**

| **Описание индекса** | **Имя** индекса |
|---|---|
| Requirement Satisfied (Удовлетворенное требование) | REQSAT.IDX |
| Test Identifier (Идентификатор теста) | TSTID.IDX |
| User ID (Идентификатор пользователя) | UID.IDX |
| Pass (Пройден) | PASS.IDX |
| Fail (Отказ) | FAIL.IDX |
| Blocked (Заблокирован) | BLCKD.IDX |

## Ссылки

Ссылки отсутствуют

## Приложение 1 —Список аббревиатур

API—Application Programming Interface (Интерфейс программирования приложений)
ASCII — American Standard Code for Information Interchange (Американский стандартный код обмена информацией)
CDR — Compact Disc Recordable (Записываемый компакт-диск)
HTML — Hypertext Markup Language (Язык гипертекстовой разметки)
ISO — International Organization for Standardization (Международная организация по стандартизации)
PPC — Power PC (Процессор Power PC)
RISC — Reduced Instruction Set Computing (Сокращенный набор вычислительных инструкций)
SQL-—Structured Query Language (Язык структурированных запросов)
SPARC — Scalable Processor Architecture (Масштабируемая процессорная архитектура)
TCL — Tool Command Language (Инструментальный командный язык)
TMT — Test Management Toolkit (Набор инструментальных средств управления тестированием)
X86 — Процессоры серии Intel

## Приложение 2 — Определение терминов

Отсутствует

## Приложение 3 — Адреса электронной почты утверждающих лиц

***От: ЧакД. Клуш (Chuck D. Klout) [cdklout@tmtco]***

Отправлено: Среда 9/7/01 14:23
Кому: Крис Браун (Chris Brown) [cbrown@tmtco]; development@tmtco
Копия: marketing@tmtco; customersupport@tmtco
Тема: Определения требований к приложению TMT, версия 1.0

Уважаемые члены команды,

Я просмотрел определения требований для первой версии приложения TMT и пришел к выводу, что этот документ весьма точно соответствует требованиям, выдвинутым нашими заказчиками. Я одобряю определения требований к приложению TMT (код TMT-RD-10) в том виде, в котором они были составлены.

Я должен собираюсь встретиться со своими заказчиками на будущей неделе с целью уточнения списка обязательного оборудования. Я также хочу получить своевременный ответ, чтобы выполнить пересмотр плана тестирования.

Заранее благодарен, Чак

Чак Д. Клут
Директор отдела маркетинга
TMTCO
cdklout@tmtco.com

***От: Сьюзи Перл (Suzie Perl [spent@tmtco.com]***

Отправлено: Четверг 9/9/2001 09:30
Кому: Крис Браун (Chris Brown) [cbrown@tmtco]; development@tmtco
Копия: marketing@tmtco; customersupport@tmtco
Тема: Определение требований TMT 1.0

Привет всем,

Хорошая работа! Я просмотрела определение требований к приложению TMT-RD-10 (версия 8) и одобрила его использование в процессе разработки в том виде, в каком оно было записано.

С наилучшими пожеланиями,
Сьюзи

Сюзанна Перл
Менеджер, отдел программирования
TMTCO

***От: Брит Гейтер (Bret Gater) [bgater@tmtco.com]***

Отправлено: Четверг 9/9/2001
Кому: Крис Браун [cbrown@tmtco]; test@tmtco; development@tmtco
Копия: marketing@tmtco; costumersupport@tmtco
Тема: Определение требований TMT 1.0

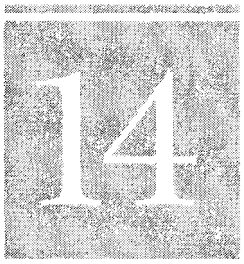
Уважаемые члены команды,

Я просмотрел определение требований TMT-RD-10 (версия 8) и одобрил его использование командой тестирования в том виде, в котором он написан.

С наилучшими пожеланиями,
Брит

Брит Гейтер
Менеджер, отдел программирования

# Пример плана тестирования




В главе 3 отмечалось, что эффективное планирование тестирования является ключевым фактором успеха проекта, оцениваемого в терминах соответствия графику и бюджету. Для удобства на рис. 14.1 повторно приводится диаграмма из главы 3, которая отражает действия, выполняемые во время составления планов тестирования.

Исходными данными для процесса планирования являются документы описания требований, которые были подробно описаны в главе 2. В главе 13 приводится достаточно реальный пример документа определения требований. Результатом выполнения действий, связанных с планированием, является план тестирования. Этот план представляет собой документ или набор документов, которые должны периодически просматриваться командами тестировщиков, разработчиков и менеджерами программных продуктов. В плане тестирования идентифицируются ресурсы, которые необходимо привлечь для тестирования программного продукта, определяются цели тестирования, методика его проведения, а также указывается, какие результаты и промежуточные продукты должны подвергаться тестированию. Содержимое и формат плана тестирования обсуждаются в главе 3, а пример план тестирования предлагается в данной главе.

Полезные рекомендации по разработке плана тестирования содержатся в документе *IEEE Standard 829, IEEE Standard for Software Test Documentation* (дополнительные сведения вопросу можно найти в главе 3). Конечно, можно воспользоваться и альтернативным форматом, но при этом не следует забывать относительно включения всех необходимых документов, диктуемых стандартом IEEE. В этой главе в качестве основы для примера плана тестирования был выбран стандарт IEEE Standard 829.

Структура плана тестирования, определяемая стандартом, включает в себя 16 компонент:

1. Идентификатор плана проведения испытаний
2. Введение
3. Компоненты, которые должны тестироваться
4. Характеристики и свойства, которые должны тестироваться
5. Характеристики и свойства, которые не должны тестироваться
6. Подход
7. Критерий успешных и неудачных испытаний
8. Критерий приостановки испытаний и требования возобновления испытаний
9. Выходные результаты тестов
10. Задачи тестирования

11. Требования окружающей среды
12. Распределение ответственности
13. Подбор кадров и подготовка персонала
14. График работ
15. Риски и непредвиденные обстоятельства
16. Утверждение плана проведения испытаний.


*Рис. 14.1. Действия, выполняемые при составлении планов тестирования.*

При внимательном изучении приведенного выше списка можно заметить, чем *tie* является стандартный план тестирования. Он не суть детализованная спецификация, описывающая методику выполнения тестов, а также не место фиксации результатов тестирования. В некоторых организациях, занимающихся тестированием, с содержимым плана комбинируются один и большее количество указанных выше пунктов. В результате появляется возможность собрать и сохранить всю документацию в одном месте. Подобный подход вполне оправдан. Тем не менее, в этой книге рассматривается модульный набор документов, который в дальнейшем подпадает под стандарт.

В оставшейся части главы предлагается пример плана тестирования, подготовленный для вымышленного набора инструментальных средств управления тестированием (Test Management Toolkit, TMT). В качестве исходных данных для плана тестирования служит документ определения требований к продукту TMT, который был рассмотрен в главе 13.

**Идентификатор документа:** TMT-TP-10
**Версия: 0.8**
**Авторы:** Крис Браун (Chris Brown)
Джеймс Варне (James Barnes)

Набор инструментальных средств управления тестированием

Версия 1.0

# План тестирования

## Хронология версий

| Версия | Дата | Автор | Описание |
|---|---|---|---|
| 0.1 | 09/02/2001 | Крис Браун | Эскиз плана тестирования для документа определения требований: TMT-RS-05 |
| 0.2 | 09/03/2001 | Крис Браун | Доводка и синхронизация с: TMT-TS-05 |
| 0.3 | 09/03/2001 | Крис Браун | Начато построение детализованных тестовых случаев |
| 0.4 | 09/05/2001 | Крис Браун | Построение тестовых случаев завершено. Окончательная доводка |
| 0.5 | 09/06/2001 | Крис Браун | Основная доводка и очистка, дополнительные тестовые случаи, заключительный формат |
| 0.6 | 09/07/2001 | Крис Браун | Заключительная синхронизация с TMT-RS-07, добавлены тесты многопользовательского режима |
| 0.7 | 09/09/2001 | Крис Браун | В раздел 2 добавлены тестовые случаи для многопользовательского режима |
| 0.8 | 09/10/2001 | Джеймс Варне | Конфигурации тестов, оценка трудозатрат и календарный график |

## Утверждено

| Отдел | | Дата утверждения |
|---|---|---|
| Маркетинга | Чак Д. Клут (Chuck D. Klout), директор отдела маркетинга | 09/11/2001 |
| Программирования | Сюзанна Перл (Suzanna Perl), менеджер отдела программирования | 09/12/2001 |
| Тестирования | Брит Гейтер (Bret Gater), менеджер отдела тестирования | 09/12/2001 |

# Содержание



## 1. Введение

Назначение этого документа состоит в детализации процедур тестирования, которые должны аттестовать функциональные возможности программного продукта TMT. Свойства, на которые производятся ссылки в этом документе, находятся в документе определения требований, именуемого "Набор инструментальных средств управления тестированием, Определение требований". Документ, определяющий требования, имеет идентификатор TMT-RD-10 и находится под управлением системы контроля документов на Web-сайте:

http//www.tmtcointernal.com/usr/www/docstores/design/reauirements/TMT-RD-10.doc

## 2. Тестируемые элементы

Ниже приводится перечень высокоуровневый список компонентов продукта, на которые имеются ссылки в этом план тестирования:

- **Тестируемая версия** — Этот элемент связан с функциональными возможностями программного продукта TMT версии 1.0.
- **Исправление ошибок** — Это первая версия профаммного продукта, поэтому в ней отсутствуют исправления ошибок, которые найдены в предыдущих версиях, требующих тестирования. Все найденные и исправленные в ходе тестирования ошибки должны быть верифицированы.
- **Носитель, на котором распространяется продукт**— Начальная версия профаммного продукта может выгружаться из Web-сайта разработчиков. Заказчики, заинтересованные в приобретении этого продукта, могут получать его на CD-ROM. Тестированию должны подвергаться оба типа распространения.
- **Документы для конечного пользователя** — Предполагается, что клиент и сервер находятся в различных местах, поэтому требуются два отдельных модуля, для каждого из которых должна предусматриваться собственная программа установки. Документы для конечного пользователя, такие как руководство пользователя, руководство по установке и примечания к версиям, могут выгружаться отдельно, чтобы заказчик мог иметь возможность просматривать системные требования и процедуры установки. Для достижения приемлемого уровня точности должно выполняться тестирование процессов установки и создания пакетов, а также просматриваться соответствующая документация.

## 3. Свойства, которые должны тестироваться

Для того чтобы удостовериться в том, что программный продукт TMT удовлетворяет требованиям, указанным в спецификации требований TMT, необходимо протестировать следующие требования:

- Требование 3.1.1. Пользовательский интерфейс
- Требование 3.1.2. Навигация
- Требование 3.1.3. Аутентификация пользователей — клиент
- Требование 3.1.4. Аутентификация пользователей — администратор
- Требование 3.1.5. Текущие проекты
- Требование 3.1.6. Завершенные проекты
- Требование 3.1.7. Создание нового проекта
- Требование 3.1.8. Изменение проекта
- Требование 3.1.9. Удаление проекта
- Требование 3.1.10. Создание тестового случая или набора
- Требование 3.1.11. Изменение тестового случая или набора
- Требование 3.1.12. Удаление тестового случая или набора
- Требование 3.1.13. Отображение теста
- Требование 3.1.14. Отображение тестового набора
- Требование 3.1.15. Прогон одиночного теста
- Требование 3.1.16. Прогон тестового набора
- Требование 3.1.17. Создание списка прогона
- Требование 3.1.18. Выполнение списка прогона
- Требование 3.1.19. Сводный отчет по ошибкам
- Требование 3.1.20. Результаты тестирования — одиночный тест
- Требование 3.1.21. Результаты тестирования — тестовый набор или список прогона

Требование 3.1.22. Создание матрицы прослеживаемости

Требование 3.1.23. Резервное копирование тестовых случаев

Требование 3.1.24. Резервное копирование тестовых наборов

Требование 3.1.25. Резервное копирование результатов прогона тестов

Требование 3.1.26. Восстановление тестовых случаев

Требование 3.1.27. Восстановление тестовых наборов

Требование 3.1.28. Восстановление результатов прогона тестов

Требование 3.1.29. Экспорт тестовых случаев

Требование 3.1.30. Экспорт тестовых наборов

Требование 3.1.31. Экспорт результатов прогона тестов

Требование 3.1.32. Получение справки

Требование 3.1.33. Многопользовательские функциональные возможности

## 4. Свойства, которые не должны тестироваться

Ниже приводится список функциональных свойств и/или конфигураций системы, которые не должны тестироваться.

- В план тестирования не включается описание функциональных возможностей и процесса установки реляционной базы данных. Предполагается, что база данных установлена и функционирует. Также предполагается, что структура данных точно определена и содержит обязательные поля с типами и размерностью, которые определены в спецификации требований. Эти требования подробно излагаются в руководствах по подготовке и установке.
- В плане не предусматривается непосредственное тестирование Web-сервера (Apache или IIS).
- План тестирования не предполагает интенсивного расширенного тестирования архитектуры клиент/сервер. Функциональные возможности, обеспечивающие работу в многопользовательском режиме, тестируются через работу пяти реальных пользователей, что определено минимальной многопользовательской конфигурацией.

## 5. Применяемый подход

Подход, предполагающий всеобъемлющее тестирование, включает тестирование свойств, регрессионное тестирование, тестирование процесса установки продукта, резервного копирования и восстановления, а также тестирование графического интерфейса пользователя. В этом разделе подробно описывается каждый упомянутый вид тестирования.

### *5.1. Тестирование свойств*

Все свойства, описанные в определении требований TMT-RD-10 должны тестироваться на выбранных комбинациях конфигураций клиент/сервер, описанных в разделе 10. Тестирование свойств предполагает функциональное и отрицательное тестирование (попытка выполнения операций и ввода данных, не предусмотренных разработчиками).

### *5.2. Регрессионное тестирование*

Поскольку это первая версия программного продукта, отсутствует потребность в верификации на предмет проявления ошибок, устраненных в предыдущих версиях. Данная версия программы отличается тем, что ошибки, исправленные на этапе системного тестирования, не разрушают ранее работоспособные функциональные возможности.

Для регрессионного тестирования первой версии программного продукта предлагается следующий подход:

- Исправление ошибок должно осуществляться по мере их обнаружения. Для каждой программной сборки, переданной в команду тестировщиков, должны прогоняться тесты, что обеспечивает гарантию того, что устраненные ошибки не проявятся снова. Другими словами, в сборке должно проверяться каждое исправление ошибки.
- Если программный продукт функционирует устойчиво, а тестовые случаи прошли успешно, перед выполнением просмотром готовности должен быть выполнен последний проход регрессионного тестирования.

### *5.3. Установка продукта*

Каждая программная сборка, переданная команде тестировщиков, устанавливается в соответствии с процедурой установки, которую будет использовать заказчик. Однако для каждой сборки модули клиента и сервера устанавливаются только на подмножестве всех возможных комбинаций платформ и операционных систем, которые указаны в спецификации требований. Предполагается, что успешная установка на одной платформе UNIX создает прецедент для успешной установки для всех остальных UNIX-подобных платформ. То же справедливо и для платформ Windows. Этот подход утвержден у заказчика (см. сообщение электронной почты утверждающего лица из отдела маркетинга).

### *5.4. Резервное копирование и восстановление*

Функционирование резервного копирования и восстановления тестируется для проектов, тестовых случаев, тестовых наборов и результатов прогона тестов. При этом должны использоваться как физические, так и логические устройства, которые являются автономными либо сетевыми. Сетевое резервное копирование является наиболее предпочтительным сценарием для заказчика, поэтому ему будет уделяться повышенное внимание.

### 5.5. *Тестирование графического интерфейса пользователя*

При тестировании графического интерфейса продукта TMT используется следующий подход:

- Графический интерфейс пользователя тестируется в браузерах Netscape Navigator и Microsoft Internet Explorer. При этом должен быть просмотрен полный состав интерфейса, а также протестированы возможности навигации в обоих браузерах.
- Все действия по тестированию выполняются в ручном режиме.
- Все дефекты отслеживаются и устраняются с помощью корпоративной системы отслеживания дефектов. Такой подход предполагает нахождение недоработок в графическом интерфейсе пользователя в ходе проведения различных оценок после завершения работы над проектом.

## 6. Критерий успешных и неудачных испытаний

Критерий успешных и неудачных испытаний для каждого тестового случая описывается через ожидаемые результаты. Если после прогона тестового случая получен ожидаемый результат, значит, тест пройден успешно. Если же после прогона теста ожидаемый результат не получен, считается, что тест потерпел неудачу. Если же тест не может быть прогнан вследствие блокирующей ошибки в сборке, результат тестирования именуется "заблокированным".

Для того чтобы продукт TMT смог успешно пройти фазу системного тестирования, 100% тестов из данного плана тестирования должны выполниться, по крайней мере, на одной программной сборке. 100% всех прогнанных тестов должны завершиться успешно, а по завершении тестирования не должна остаться неустраненной ни одна серьезная ошибка.

## 7. Критерий приостановки испытаний и требования возобновления испытаний

Если хотя бы одна фундаментальная функциональная возможность оказывается неработоспособной, например, установка и запуск программы, тестирование должно быть приостановлено до тех пор, пока соответствующая функциональность не станет доступной. Поиск катастрофических ошибок

должен продолжаться, если только обнаруженные ошибки не столь серьезны и не заблокировано 50% и более тестовых случаев. Если тестирование оказывается приостановленным, команды разработчиков и тестеров должны ежедневно встречаться с целью достижения прогресса в возобновлении испытаний. Встречи продолжаются до тех пор, пока не будет достигнуто согласие возобновить процесс тестирования.

## 8. Выходные результаты тестов

Перечисленные ниже элементы представляют собой рабочие продукты, которые появляются в результате выполнения тестирования:

- Данный план тестирования.
- Матрица прослеживаемое™ требований.
- Документ со спецификациями тестов.
- Отчеты по результатам прогона тестов.
- Ежедневные обновления состояния тестирования, направляемые менеджерам по тестированию и разработке.
- Отчеты о дефектах (ошибках).

За примечания по версии несут ответственность разработчики; однако, примечания по версии должны просматриваться и одобряться командой тестирования до пересмотра готовности продукта.

## 9. Задачи тестирования

Ниже перечислены задачи, которые должны выполняться во время тестировании продукта TMT:

- Выполнение тестирования процесса установки продукта.
- Прогон тестов для свойств и построение отчета об ошибках.
- Верификация фактов устранения ошибок.
- Выполнение тестов резервного копирования и восстановления.
- Выполнение тестирования графического интерфейса пользователя.
- Ведение обзоров по ошибкам.
- Подготовка отчетов о состоянии тестов.
- Написание отчета по результатам тестирования.

В этом разделе оцениваются временные затраты (в человеко-часах), которые потребуются для выполнения перечисленных выше задач. Эти оценки трудозатрат основаны на прошедшем 09.01.2001 сеансе Wideband Delphi. Фактором, оказывающим наибольшее влияние на объем времени и ресурсов, которые необходимы для тестирования продукта TMT, является количество клиентских и серверных операционных систем, оговоренное в документе определения требований. Сводку по операционным системам можно найти в таблице 9.1.

**Таблица 9.1. Клиентские и серверные операционные системы**

| Клиентская операционная система | Серверная операционная система |
|---|---|
| Microsoft Windows | Microsoft Windows |
| - Windows 85 | - Windows NT 3.51 и выше |
| - Windows 98 | UNIX |
| - Windows ME | - Sun Solaris 2.6 и выше |
| - Windows XP | - HPUX 10.x и выше |
| - Windows NT 3.51 и выше | - Open BSD |
| - Windows 2000 | - AIX 2.4.1. и выше |
| Apple | - SCO Open Desktop |
| - MAC OS 9.x и выше | - Linux Red Hat 6.x и выше |

Как показано в таблице 9.1, имеется семь клиентских и семь серверных операционных систем, упомянутых в определении списка требований. При этом необходимо, чтобы использовалась только одна операционная система из списка, куда входят Windows NT, Solaris 2.6, OS9.X и Linux 6.x. Следовательно, существует 49 возможных комбинаций клиентских и серверных операционных систем. Установка каждой комбинации системы клиент/сервер требует в среднем 5 часов. Общее время, необходимое для тестирования процесса установки продукта, составляет 5 x 49 = 245 часов, или около 30 рабочих дней. Обратите внимание, что процесс предполагает установку операционной системы, приложения реляционной базы данных и приложения TMT для клиента и сервера, а также установку клиентского браузера. Если запланировать тестирование свойств и резервного копирования для всех 49 комбинаций, объемы времени, необходимого для тестирования TMT, возрастут до немыслимых размеров.

Вследствие больших временных затрат на тестирование всех возможных комбинаций, во время выполнения тестирования процесса установки продукта используют лишь подмножество комбинаций клиент/сервер, которое показано в таблице 9.2.

**Таблица 9.2. Выбранные комбинации операционных систем, применяемые при тестировании установки продукта**

| Комбинация | Клиентская операционная система | Серверная операционная система |
|---|---|---|
| **1** | - Windows 95 | - Windows NT 3.51 |
| **2** | - Windows 98 | - Sun Solaris 2.6 |
| **3** | - Windows ME | -HPUX 10.1 |
| **4** | - Windows XP | - Open BSD |
| **5** | - Windows NT 3.51 | - AIX 2.4.1 и выше |
| **6** | - Windows 2000 | - Linux 6.5 |
| **7** | - MAC OS 9.0 | - SCO Open Desktop |

Сокращение количества комбинаций должно приводить к экономии ресурсов, затрачиваемых на тестирование продукта. В данном случае на тестирование уходит 35 человеко-часов, что значительно меньше 245 часов, необходимых для проверки всех комбинаций. И все же, количество комбинаций, приведенных в таблице 9.2, остается довольно-таки большим применительно к тестированию свойств и резервного копирования/восстановления. Для тестирования свойств и резервного копиро-

вания/восстановления количество комбинаций клиент/сервер может быть сокращено до двух конфигураций (см. таблицу 9.3).

**Таблица 9.3. Выбранные комбинации операционных систем, применяемые при тестировании свойств и резервного копирования/восстановления**

| Комбинация | Клиентская операционная система | Серверная операционная система |
|---|---|---|
| 1 | - Windows 98 | - Sun Solaris 2.6 |
| 2 | - Windows 2000 | - Linux 6.5 |

Предполагая что использование выделенных комбинаций операционных систем, которые показаны в таблицах 9.2 и 9.3, в таблице 9.4 приводятся оценки трудозатрат для каждого цикла тестирования приложения TMT. Обратите внимание, что одиночный цикл тестирования включает выполнение всего набора запланированных тестов на кандидате на программную сборку. Для тестирования приложения TMT необходимо воспользоваться тремя циклами тестирования, поэтому общие трудозатраты сводятся к утроенному объему трудозатрат, перечисленных в таблице 9.4.

**Таблица 9.4. Оценки трудозатрат для каждого цикла тестирования**

| Задача | Время (часы) |
|---|---|
| Тестирование процесса установки продукта | 35 |
| Прогон тестов свойств и построение отчетов по ошибкам | 16 |
| Верификация исправлений ошибок | 24 |
| Тесты резервного копирования и восстановления | 24 |
| Тестирование графического интерфейса пользователя | 16 |
| Отчет о состоянии тестирования | 16 |
| Ведение обзоров по ошибкам | 8 |
| Итого | 139 |

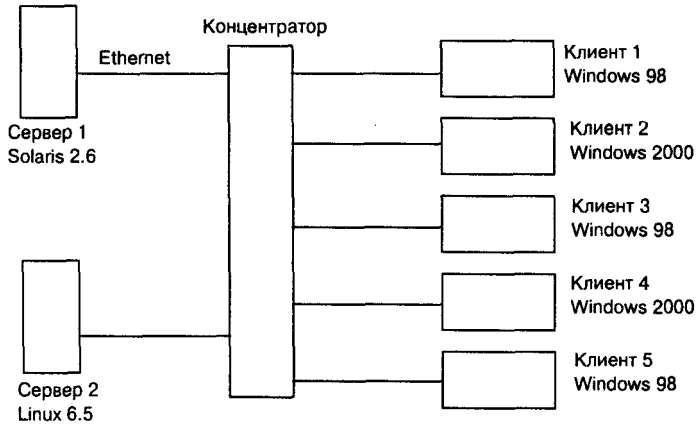



*Рис. 10.1. Тестовая конфигурация # 1.*

## 10. Конфигурации тестов

Диаграмма для тестовой конфигурации #1, показанная на рис. 10.1, будет использоваться во время тестирования свойств, резервного копирования/восстановления и графического интерфейса пользователя. В этой тестовой конфигурации присутствуют два сервера и пять клиентов, причем серверы и клиенты подключены к общей сети. Тестовая конфигурация #1 основывается на выделенных комбинациях операционных систем клиент/сервер, которые приведены в таблице 9.3. Предполагается, что тестовая конфигурация #1 будет первичной конфигурацией, используемой во время верификации ошибок.

Несмотря на то что на рис. 10.1 это явно не показано, все серверы и клиенты выполняют приложение баз данных Oracle как систему реляционных баз данных, которая задействована в продукте TMT. Это ограничение тестовой конфигурации должно быть включено в примечания по версии. Предполагается, что при будущем тестировании будут использоваться другие системы управления базами данных.

Также на рисунке явно не указано, что в качестве приложения браузера для клиентов 1, 3 и 5 выбран Internet Explorer, а для клиентов 2 и 4 - Netscape Navigator. На рис. 10.2 показана диаграмма для тестовой конфигурации #2. Эта конфигурация предназначена для тестирования процесса установки продукта, но при необходимости может использоваться и для другого тестирования. Обратите внимание, что тестовая конфигурация #2 является единственной, которая поддерживает тестирование на платформе Apple Macintosh.

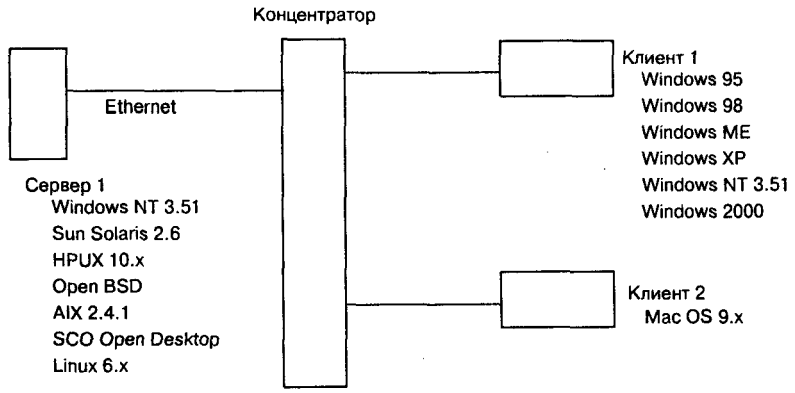



*Рис. 10.2. Тестовая конфигурация #2.*

### 11. Распределение ответственности

В этом разделе обсуждается ответственность за тестирование на организационном уровне; здесь не рассматриваются роли и ответственность отдельных инженеров по тестированию, поскольку это служит материалом следующего раздела.

*Ответственность команды разработки*

- Объединение тестируемых свойств на этапе их создания.
- Выполнение комплексных испытаний на свойствах перед их упаковкой в форме сборки для команды тестирования.
- Подготовка программных сборок для команды тестирования в сроки, устанавливаемые на еженедельных собраниях команд разработки и тестирования.
- Помощь тестировщикам в обосновании результатов тестирования и, при необходимости, в характеризации ошибок, чтобы в систему отслеживания дефектов вводились точные отчеты.
- Устранение ошибок, информация о которые введена в систему отслеживания дефектов.

- Подготовка примечаний по версии для тех ошибок, которые не устранены в версии, поставленной заказчику.
- Реализация обходного пути "малого влияния на заказчика" для тех ошибок, которые не были устранены, но упоминаются в примечании по версии.

*Ответственность команды тестирования*

- Прогон запланированных тестов и ввод сообщений об ошибках в систему отслеживания дефектов в случае получения аномальных результатов.
- Помощь разработчикам в воспроизведении ошибок.
- Ведение регулярных обзоров ошибок на этапе системных испытаний. Обзоры должны составляться еженедельно, если только команды тестирования и разработки не придут к выводу, что отчеты необходимо составлять чаще.
- Проверка успешности устранения ошибок, которая позволяет удостовериться в том, что исправленное программное обеспечение демонстрирует ожидаемое поведение (удовлетворяет определенным требованиям).
- Подготовка еженедельных отчетов о состоянии процесса тестирования и обнаружении ошибок.
- Подготовка отчета по результатам прогона тестов в конце этапа системных испытаний. Резюме этого отчета будет присутствовать в обзоре степени готовности продукта. Резюме должно включать в себя определение, достигнут ли критерий завершения тестирования, обозначенный в разделе 6.

## 12. Подбор кадров и подготовка персонала

Роли и ответственности персонала во время тестирования приложения TMT показаны в таблице 12.1.

Д. Нгуен (D. Nguyen) должен обучиться администрированию баз данных Oracle, а К. Тейлор (C. Taylor), новичок в компании, должна пройти внутренние курсы обучения по тестированию программного обеспечения "Software Testing 101", руководимые Дж. Барнсом (J.Barnes). Все обучение должно завершиться до передачи разработчиками первой программной сборки в команду тестирования.

**Таблица 12.1. Персонал тестеров**

| **Сотрудник** | Должность | Ответственность | Доступность |
|---|---|---|---|
| Дж. Варне | Руководитель отдела тестирования | Координация тестирования, построение отчетов и тестирование свойств | 100 % |
| Д. Нгуен | Тестировщик | Установка продукта и платформы тестирования | 100% |
| С. Скефиди | Тестировщик | Тесты резервного копирования/восстановления | 50% |
| К. Тейлор | Тестировщик | Тесты графического интерфейса пользователя | 50% |
| Б. Вилер | Разработчик | Тестирование модулей комплексные испытания, исправление ошибок | 100 % |

## 13. Календарный график

Календарный график системного тестирования продукта TMT показан в таблице 13.1. Испытание герметичности, показанное на графике, выполняется во время предварительного знакомства с программным обеспечением. В течение этого испытания обеспечивается обнаружение в приложении основных блокирующих ошибок, а также дефектов в конфигурациях тестов и тестовых случаях. Испытание герметичности должно состоять из подмножества тестовых случаев установки и свойств.

**Таблица 13.1. Календарный график тестирования продукта TMT**

| Задача | Дата начала | Дата завершения |
|---|---|---|
| Установка и отладка тестовых конфигураций #1 и #2 | 10/1 | 10/12 |
| Испытание герметичности на предварительной сборке | 10/15 | 10/19 |
| Цикл тестирования #1 | 10/22 | 11/2 |
| Цикл тестирования #2 | 22/5 | 11/16 |
| Цикл тестирования #3 | 11/19 | 11/30 |
| Пересмотр степени готовности | 12/3 | 12/3 |

Предполагается, что тестирование модулей и комплексные испытания были реализованы до установки тестовой конфигурации #1 и все катастрофические ошибки, обнаруженные в ходе этого тестирования, устранены до начала стадии системного тестирования.

## 14. Риски и непредвиденные обстоятельства

В таблице 14.1 перечислены риски, связанные с процессом тестирования продукта TMT, вместе с оценочными значениями вероятностей их возникновения, влиянием и кратким описанием плана смягчения этого влияния.

**Таблица 14.1. Идентификация рисков и меры по смягчению их воздействия**

| Риск | Вероятность | Влияние | Смягчение влияния |
|---|---|---|---|
| Написание тестовых случаев для процесса резервного копирования скорее всего задержится, поскольку разработка подсистемы резервного копирования еще не завершена, а созданный прототип выявил серьезные проблемы. | **75%** | Основное | Руководитель команды тестирования должен постоянно принимать участие во встречах разработчиков и по мере получения новой информации пересматривать тестовые случаи. |
| К. Тейлор может отсутствовать на работе в течение всего ноября по семейным обстоятельствам. Ее отсутствие привело бы к смещению графика на 2 дня в каждом тестовом цикле. | **50%** | Незначительное | Один из разработчиков графического интерфейса, Р. Карини (R. Carini), согласился выполнить его тестирование, а С. Скефиди должен подготовить обзор результатов прогона тестов, если Тейлор будет отсутствовать. |
| Контракт на обслуживание сети завершился, а его возобновление находится на стадии переговоров. Если контракт не будет возобновлен до начала тестирования, то календарный график может не выполняться из-за возможных проблем с сетью. | **20%** | Незначительное | Необходимо отслеживать состояние контракта на обслуживание и обратить внимание руководства на возможный риск, если соглашение не будет достигнуто до начала испытаний герметичности. |

## Ссылки

*Chris Brown, Test Management Toolkit, Requirements Definition (Набор инструментальных средств управления тестированием, Определение требований).* Документ TMT-RD-10, который размещается под управлением системы контроля документов по адресу:

http://www.tmtcointemal.corn/usr/www/docstores/desian/reauirements/TMT-RD-10.doc

## Приложение 1 — Список аббревиатур

API — Application Programming Interface (Интерфейс программирования приложений)
ASCII — American Standard Code for Information Interchange (Американский стандартный код обмена информацией)
CDR — Compact Disc Recordable (Записываемый компакт-диск)
HTML — Hypertext Markup Language (Язык гипертекстовой разметки)
ISO — International Organization for Standardization (Международная организация по стандартизации)
PPC — Power PC (Процессор Power PC)
RISC — Reduced Instruction Set Computing (Сокращенный набор вычислительных инструкций)
SQL — Structured Query Language (Язык структурированных запросов)
SPARC — Scalable Processor Architecture (Масштабируемая процессорная архитектура)
TCL — Tool Command Language (Инструментальный командный язык)
TMT — Test Management Toolkit (Набор инструментальных средств управления тестированием)
X86 — Процессоры серии Intel

## Приложение 2 — Определение терминов

Отсутствует

## Приложение 3 — Сообщения электронной почты от утверждающих лиц

***От: ЧакД. Клуш (Chuck D. Klout) [cdklout@tmtco]***

Отправлено: Среда 9/11/01 16:40
Кому: Крис Браун (Chris Brown) [cbrown@tmtco]; development@tmtco
Копия: marketing@tmtco; customersupport@tmtco
Тема: План тестирования TMT, версия 1.0

Уважаемые члены команды,

Во-первых, я хочу выразить благодарность команде за напряженную работу по написанию требований и плана тестирования для данной версии программного продукта. Я утверждаю план тестирования TMT-TP-10, версия 8, в том виде, в котором он написан.

Я встречался с нашими заказчиками для определения обязательного перечня сертифицированного оборудования для данной версии программного продукта. Утверждения, проведенные в плане тестирования в связи с предполагаемой совместимостью с ограниченным числом комбинаций операционных систем, имеют смысл. В связи с этим мы можем продолжать работы с сокращенным списком оборудования и операционных систем, на который имеются ссылки в разделах 9 и 10 при рассмотрении конфигураций тестов. Такая комбинация аппаратного и программного обеспечения покрывает и те конфигурации, которые заказчики используют в настоящее время, так и те, которые планируется установить в будущем.

Спасибо,
Чак

Чак Д. Клут
Директор отдела маркетинга
TMTCO
cdklout@tmtco.com

***От: Сьюзи Перл (Suzie Perl [spent@tmtco.com]***

Отправлено: четверг 9/12/2001 09:30
Кому: Крис Браун (Chris Brown) [cbrown@tmtco]; development@tmtco
Копия: marketing@tmtco; customersupport@tmtco
Тема: План тестирования TMT 1.0

Привет всем,

Хорошая работа! Я просмотрела план тестирования TMT-TP-10, версия 8, для первой версии TMT и утверждаю его в том виде, в котором он написан.

С наилучшими пожеланиями,
Сьюзи

Сюзанна Перл
Менеджер, отдел программирования
TMTCO

***От: Брит Гвйтер (Bret Gater) [bgater@tmtco.com]***

Отправлено: четверг 9/12/2001 7:30
Кому: Крис Браун [cbrown@tmtco]; test@tmtco; development@tmtco
Копия: marketing@tmtco; costumersupport@tmtco
Тема: План тестирования TMT 1.0

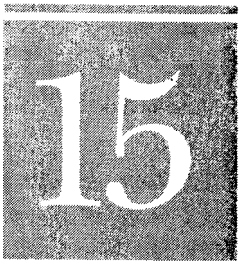
Уважаемые члены команды,

Я просмотрел план тестирования TMT-TP-10, версия 8, и утверждаю его использование командой тестирования в том виде, в котором он написан.

С наилучшими пожеланиями,
Брит

Брит Гейтер
Менеджер, отдел программирования

# Примеры проектирования и разработки тестов




В главе 4 упоминалось, что процесс создания сценариев эффективных тестовых случаев состоит из двух частей. Исходя из предположения, что непротиворечивые определения требований готовы, а план тестирования в соответствие с требованиями составлен, разработка тестовых случаев сводится к выполнению следующих этапов:

- Проектирование тестов
- Разработка детализованных тестовых процедур
- Верификация и отладка тестовых процедур.

Этот общий подход можно применять для тестов, выполняемых как в ручном, так и в автоматическом режимах; реально получается так, что сначала лучше разработать и отладить процедуры вручную, а затем добавить возможности автоматизации.

На рис. 15.1 показана диаграмма, отображающая действия по проектированию и разработке тестов. Первичными входными данными для процесса разработки является набор документов, описывающие план тестирования, который обсуждался в главе 3. В плане тестирования предлагается подход, а также определяется круг задач, которые должны быть решены в ходе тестирования. Потребуется определить архитектуру тестов, что означает определение, по меньшей мере, тестовых наборов. Кроме того, необходимо будет определить набор тестовых конфигураций, на которые будут ориентироваться создаваемые тесты. Пример плана тестирования был представлен в предыдущей главе.

Результатом действий по проектированию и разработке тестов является набор просмотренных и отлаженных тестовых случаев, готовых для использования в системных и приемочных испытаниях. Тестовые случаи должны отображаться на требования, сформулированные заказчиком. Они должны обеспечивать хорошее покрытие за счет тестирования, по меньшей мере, всех высокоприоритетных требований, а в идеале — абсолютно всего перечня требований. Тестовые случаи также должны осуществлять хорошее покрытие кода путем прохода большинства, если не всех, логических ветвей кода.

Если в распоряжение специалиста по тестированию поступают спецификация требований и план тестирования, он может приступить к проектированию тестов. Этот процесс связан с подготовкой следующих действий:

- Определение целей тестирования
- Определение входных данных, необходимых для прогона каждого теста

- Определение конфигурации, необходимой для каждого теста
- Проведение обзора проектов, что обеспечивает техническую точность, а также полное покрытие тестами всех требований.

Все перечисленные этапы подробно рассматриваются в главе 4. По завершении процесса проектирования тестов появляются два рабочих документа: документ по проектам тестов и документ по спецификации тестов. Эти документы можно скомбинировать, как показано в примере, представленном в данной главе.

Назначение документа, включающего описание проектов тестов, состоит в охвате абсолютно всей информации, которая получается в результате выполнения действий по проектированию тестов. Этот документ может принимать форму электронной таблицы, таблицы, генерируемой текстовым процессором, или же форму базы данных. Он может быть результатом, который генерирует инструментальное средство автоматизации управления требованиями и тестами, причем полученные в таком случае тесты основываются на требованиях. В примере документа, содержащего спецификации тестовых процедур, который представлен далее в главе, проекты тестов были сгенерированы при помощи Microsoft Excel и затем помещены в документ описания спецификаций.


*Рис. 15.1. Проектирование и реализация тестов*

Процесс проектирования тестов в рассматриваемом примере тесно завязан на формат, определенный в главе 4, который для удобства приводится на рис. 15.2.

Рассматриваемая в данной главе таблица имеет сходство с матрицей прослеживаемости требований (Requirements Traceability Matrix, RTM), которая обсуждалась в

главе 2 (см. рис. 2.5). Первые два столбца таблицы должны соответствовать записям в RTM-матрице, а остальные столбцы связаны с данными, которые рассматривались в предыдущих трех разделах этой главы. Из-за подобия таблиц разработок и RTM-матрицы пример самой матрицы не приводится.

Непосредственно после создания документа с проектами тестов необходимо пересмотреть связанные с ним материалы, т.е. документ определения требований, определения входных данных для теста и конфигурации самих тестов. Как упоминалось в главе 4, назначение такого обзора связано с выполнением следующих верификаций:

- Верификация на предмет того, что при составлении документа проекта тестов учтены все требования. Если требование не тестируется, в RTM-матрице или в документе по разработке теста должна появиться соответствующая запись, описывающая причину, почему данное требование не тестируется.
- Верификация на предмет того, что с каждым тестовым случаем связан соответствующий набор входных данных.
- Верификация на предмет того, что с каждым тестовым случаем связана подходящая конфигурация, а тестовые конфигурации не являются избыточными.

Документ с проектами тестов служит основанием для создания детализированных тестовых процедур. Как указано в обзоре, детализированную разработку тестов можно поручить специалистам команды. Примеры детализованных тестовых процедур содержатся в документе, озаглавленном "Спецификация тестовой процедуры", который можно найти в конце главы.

| Идентификатор определения требования | Идентификатор системного тестового случая | Входные данные теста | Конфигурация теста | Цель теста |
|---|---|---|---|---|
| RD2.2.1 | ST2.2.4 | TP_Input1.doc | TC2.0 | T02.2.4 Верификация на предмет того, что квалифицированный пользователь может выбрать и просмотреть любой созданный и сохраненный документ с планом тестирования. |

***Рис. 15.2. Пример записи в документе с проектами тестов***

Детализованные тестовые процедуры, рассматриваемые **в** примере, записаны в виде пар "действие/ожидаемый результат". Это означает, что тестировщик должен выполнять определенные действия, такие, например, как щелчок на элементе меню и сравнение результата этого действия с ожидаемым. Если ожидаемый результат получен, тест считается пройденным; в противном случае результат прогона теста рассматривается как неудачный.

С целью экономии пространства тестовые случаи не записываются в формате шаблона тестовых случаев, который рассматривался в главе 4. Тем не менее, дабы

подчеркнуть факт сохранения идей, лежащих в основе шаблона тестового случая, последний еще раз приводится на рис. 15.3.

Основными элементами шаблона тестового случая являются:

- Идентификатор тестового случая — включает номер версии.
- Владелец теста — имя и фамилия того, кто поддерживает тест (это не всегда автор теста).
- Дата создания последней версии — помогает уточнить, является ли версия теста актуальной.
- Имя теста — описательное название теста, которое упрощает его поиск и понимание его содержания. Не рекомендуется использовать имена, не имеющие смысловой нагрузки, наподобие "xxxLLL0123.tst".
- Расположение теста — полный путь, включая имя сервера.
- Тестируемое требование — должен указываться уникальный идентификатор требования из документа описания требований.
- Цель тестирования — краткое и ясное изложение цели, которая достигается данным тестом. Более подробную информацию можно найти в разделе "Определение целей теста" главы 4).
- Конфигурация теста — спецификации входных и выходных данных, а также описание среды тестирования.
- Установка теста — по аналогии с процедурой тестирования, описываются действия, которые должен выполнять тестировщик, и ожидаемые результаты. Если процесс установки автоматизирован, то сама установка может принимать форму единственной команды, например, `run setupSC03.pi`.
- Процедура тестирования — описание действий, которые должен выполнить тестировщик, и ожидаемый результатов.
- Взаимозависимости тестовых случаев — определение тестовых случаев, которые необходимо выполнить перед данным тестовым случаем, чтобы достигнуть известных начальных условий.
- Очистка теста — если система переводится в нестабильное состояние или же ей передаются преднамеренно запорченные данные, то при помощи очистки можно выполнить откат.

| **Информация о тестовом случае** | | | |
|---|---|---|---|
| **Идентификатор тестового случая** | SC03 ver3.0 | | |
| **Владелец теста** | Джин Дуглас (Jean Douglas) | | |
| **Местонахождение теста (путь)** | TestServer:D:\TestProject\TestSuite\SC03.doc | | |
| **Дата последнего пересмотра** | Месяц/день/год | | |
| **Тестируемое требование** | SC101 | | |
| **Конфигурация средств тестирования** | ST02 | | |
| **Взаимозависимость тестовых случаев** | Выполнить прогон теста SC01 и его настройку перед прогоном данного теста. | | |
| **Цель теста** | Проверить, что допустимые входные значения веса отправляемого груза дают правильные значения стоимости доставки, и что недопустимые входные значения приводят к выдаче сообщений об ошибке. | | |
| **Методика тестирования** | | | |
| **Настройка на прогон теста** | Не проводится | | N/A |
| | | | |
| **Шаг** | **Действие** | **Ожидаемый результат** | **Отметка (V)** |
| 1. | Щелкнуть на элементе "Стоимость доставки" в главном меню. | Отображается меню "Стоимость доставки" | **(V)** |
| 2. | Ввести "101" в поле веса доставляемого груза | Сообщение об ошибке "Неправильно указан вес доставляемого груза" | **(V)** |
| 3. | Ввести "0" в поле веса доставляемого груза | Сообщение об ошибке "Неправильно указан вес доставляемого груза" | **X** |
| 4. | Ввести "100" в поле веса доставляемого груза | Указан вес доставляемого груза "100 унций" | **(V)** |
| 5. | Ввести "1" в поле веса доставляемого груза | Указан вес доставляемого груза "1 унция" | **(V)** |
| | | | |
| **Очистка после прогона теста** | Не производится | | **N/A** |
| **Результаты теста** | | | |
| **Тестировщик: JD** | **Дата прогона теста:** месяц/день/год | **Результат теста (P/F/B):** F | |
| **Примечания:**<br>- Тест потерпел неудачу на шаге 3.<br>- При возникновении неисправности выдается код ошибки BR1011. | | | |

*Рис. 15.3. Пример тестового случая*

**Идентификатор документа:** TMT-TPS-10
**Версия: 0.5**
**Авторы:** Крис Браун (Chris Brown)
Джеймс Барнс (James Barnes)

Набор инструментальных средств управления тестированием

Версия 1.0

# Спецификация тестовой процедуры

## Хронология версий

| Версия | Дата | | Описание |
|---|---|---|---|
| 0.1 | 08/29/2001 | Джеймс Барнс | Проекты тестов |
| 0.2 | 09/03/2001 | Крис Браун | Начато создание детализованных тестовых случаев |
| 0.3 | 09/05/2001 | Крис Браун | Завершено создание тестовых случаев |
| 0.4 | 09/06/2001 | Крис Браун | Основная доводка после пересмотра тестовых случаев |
| 0.5 | 09/10/2001 | Джеймс Барнс | Обсуждение тестов графического интерфейса и процесса установки |

## Утверждено

| | | Дата утверждения |
|---|---|---|
| Программирования | Сюзанна Перл (Suzanna Perl), менеджер отдела программирования | 09/11/2001 |
| Маркетинга | Чак Д. Клут (Chuck D. Klout), директор отдела маркетинга | 09/12/2001 |
| Тестирования | Брит Гейтер (Bret Gater), менеджер отдела тестирования | 09/12/2001 |

## Содержание

## 1. Введение

Целью создания этого документа является подробное описание тестовых процедур, разработанных для аттестации функциональных возможностей программного продукта, именуемого набором инструментальных средств управления тестированием (Test Management Toolkit, TMT). Свойства, на которые производятся ссылки в этом документе, находятся в документе определения требований, именуемого "Набор инструментальных средств управления тестированием, Определение требований". Документ, определяющий требования, имеет идентификатор TMT-RD-10 и находится под управлением системы контроля документов на Web-сайте:

httD//www.tmtcointernal.com/usr/www/docstores/desian/reauirements/TMT-RD-10.doc

Разработка тестов основывается на плане тестирования TMT, который находится в документе TMT-TP-10, расположенном на Web-сайте:

http//www.tm tcointernal.com/usr/www/docstores/desian/reauirements/TMT-TP-10.doc

Проекты тестов описаны в разделе 3, а тестовые процедуры представлены в разделе 5.

## 2. Свойства, которые должны тестироваться

Для того чтобы удостовериться в том, что программный продукт TMT удовлетворяет требованиям, указанным в спецификации требований TMT, необходимо протестировать следующие требования:

- Требование 3.1.1. Пользовательский интерфейс
- Требование 3.1.2. Навигация
- Требование 3.1.3. Аутентификация пользователей — клиент
- Требование 3.1.4. Аутентификация пользователей — администратор
- Требование 3.1.5. Текущие проекты
- Требование 3.1.6. Завершенные проекты
- Требование 3.1.7. Создание нового проекта
- Требование 3.1.8. Изменение проекта
- Требование 3.1.9. Удаление проекта
- Требование 3.1.10. Создание тестового случая или набора
- Требование 3.1.11. Изменение тестового случая или набора
- Требование 3.1.12. Удаление тестового случая или набора
- Требование 3.1.13. Отображение теста
- Требование 3.1.14. Отображение тестового набора
- Требование 3.1.15. Прогон одиночного теста
- Требование 3.1.16. Прогон тестового набора
- Требование 3.1.17. Создание списка прогона
- Требование 3.1.18. Выполнение списка прогона
- Требование 3.1.19. Сводный отчет по ошибкам
- Требование 3.1.20. Результаты тестирования — одиночный тест
- Требование 3.1.21. Результаты тестирования — тестовый набор или список прогона
- Требование 3.1.22. Создание матрицы прослеживаемости
- Требование 3.1.23. Резервное копирование тестовых случаев
- Требование 3.1.24. Резервное копирование тестовых наборов
- Требование 3.1.25. Резервное копирование результатов прогона тестов
- Требование 3.1.26. Восстановление тестовых случаев

- Требование 3.1.27. Восстановление тестовых наборов
- Требование 3.1.28. Восстановление результатов прогона тестов
- Требование 3.1.29. Экспорт тестовых случаев
- Требование 3.1.30. Экспорт тестовых наборов
- Требование 3.1.31. Экспорт результатов прогона тестов
- Требование 3.1.32. Получение справки
- Требование 3.1.33. Многопользовательские функциональные возможности

## 3. Проекты тестов

Применяемый подход предполагает регрессионное тестирование, а также тестирование функциональных возможностей, процесса установки, резервного копирования и восстановления и графического интерфейса пользователя. Каждый тип тестирования подробно рассматривается в данном разделе.

Проекты тестов для проверки функциональных возможностей показаны в таблице 3.1. В таблице приведены идентификаторы требований, идентификаторы тестов, входные данные и конфигурация для тестов, а также цели прогона каждого теста. Некоторые функциональные возможности, такие как экспорт данных и экраны справочной системы, вынесены за пределы основной массы тестов функциональных возможностей, поскольку в первой сборке они не реализованы. Эти возможности рассматриваются в разделе "Дополнительные возможности".

**Таблица 3.1. Разработка тестов свойств**

| **Идентификатор требования** | **Идентификатор системного тестового случая** | **Входные данные теста** | Тестовая **конфигурация** | **Цель проведения теста** |
|---|---|---|---|---|
| RD3.1.1 | TC3.1.1 | Нет | Тестовая конфигурация #1 | Проверка доступности пользовательского интерфейса TMT путем вызова главного меню |
| RD3.1.2 | TC3. 1.2 | Нет | Тестовая конфигурация #1 | Проверка доступности пользовательского интерфейса TMT путем навигации в рамках меню |
| RD3.1.3 | TC3. 1.3 | Учетные записи пользователей | Тестовая конфигурация #1 | Проверка системы безопасности доступа пользователей для допустимых и недопустимых пользователей со стороны клиента |
| RD3.1.4 | TC3. 1.4 | Учетные записи администраторов | Тестовая конфигурация #1 | Проверка системы безопасности администрирования для допустимых и недопустимых пользователей со стороны сервера |
| RD3.1.5 | TC3. 1.5 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности просмотра проектов зарегистрированными пользователями |

*Продолжение табл. 3.1*

| Идентификатор требования | Идентификатор системного тестового случая | Входные данные теста | Тестовая конфигурация | Цель проведения теста |
|---|---|---|---|---|
| RD3.1.6 | TC3. 1.6 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности просмотра завершенных проектов аутентифицированными пользователями |
| RD3.1.7 | TC3. 1.7 | Пример данных информации теста | Тестовая конфигурация #1 | Проверка возможности создания новых проектов зарегистрированными пользователями с выбором существующих тестовых случаев и наборов |
| RD3.1.8 | TC3. 1.8 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности изменения проектов зарегистрированными пользователями |
| RD3.1.9 | TC3. 1.9 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности удаления проектов зарегистрированным пользователем |
| RD3.1.10 | TC3.1.10 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя создавать одиночные тестовые случаи либо наборы |
| RD3.1.11 | TC3.1.11 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя изменять одиночные тестовые случаи либо наборы |
| RD3.1.12 | TC3.1.12 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя удалять одиночные тестовые случаи либо наборы |
| RD3.1.13 | TC3.1.13 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя отображать (просматривать) тестовые случаи, связанные с проектом |
| RD3.1.14 | TC3.1.14 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя отображать (просматривать) тестовые наборы, связанные с проектом |
| RD3.1.15 | TC3.1.15 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя прогонять тестовые случаи, связанные с проектом |

***Окончание табл. 3.1***

| Идентификатор требования | Идентификатор системного тестового случая | Входные данные теста | Тестовая конфигурация | Цель проведения теста |
|---|---|---|---|---|
| RD3.1.16 | TC3.1.16 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя прогонять тестовые наборы, связанные с проектом |
| RD3.1.17 | TC3.1.17 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности создания списка прогона для проекта |
| RD3.1.18 | TC3.1.18 | Пример данных по проекту | Тестовая конфигурация #1 | Проверка возможности выполнения списка прогона для проекта |
| RD3.1.19 | TC3.1.19 | Пример данных по ошибкам | Тестовая конфигурация #1 | Проверка возможности отображения (просмотра) зарегистрированным пользователем списка всех ошибок проекта |
| RD3.1.20 | TC3.1.20 | Пример результатов прогона теста | Тестовая конфигурация #1 | Проверка возможности отображения (просмотра) зарегистрированным пользователем результатов прогона тестового случая |
| RD3.1.21 | TC3. 1.21 | Пример результатов прогона теста | Тестовая конфигурация #1 | Проверка возможности отображения зарегистрированным пользователем результатов выполнения тестового набора или списка прогона |
| RD3.1.22 | TC3. 1.22 | Пример данных по требован иям | Тестовая конфигурация #1 | Проверка возможности создания зарегистрированным пользователем матрицы прослеживаемости требований для проекта с корректными данными, описывающими требования |

### *3.1. Регрессионное тестирование*

Поскольку это первая версия программного продукта, отсутствует потребность в верификации на предмет проявления ошибок, устраненных в предыдущих версиях. Данная версия программы отличается тем, что ошибки, исправленные на этапе системного тестирования, не разрушают ранее работоспособные функциональные возможности. Регрессионное тестирование в данном случае включает все тестовые случаи.

### *3.2. Резервное копирование и восстановление*

Функционирование резервного копирования и восстановления тестируется для проектов, тестовых случаев, тестовых наборов и результатов прогона тестов. При этом должны использоваться как физические, так и логические устройства, которые являются автономными либо сетевыми. Сетевое резервное копирование является наиболее предпочтительным сценарием для заказчика, поэтому ему будет уделяться повышенное внимание.

Проекты тестов для резервного копирования и восстановления показаны в таблице 3.2. В таблице отражены идентификаторы требований, идентификаторы тестов, входные данные и конфигурация для тестов, а также цели прогона каждого теста.

**Таблица 3.2 Разработка теста операций резервного копирования/восстановления**

| Идентификатор требования | Идентификатор системного тестового случая | Входные данные теста | Тестовая конфигурация | Цель проведения теста |
|---|---|---|---|---|
| RD3.1.23 | TC3. 1.23 | Пример тестовых данных | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя выполнять резервное копирование одиночного тестового случая |
| RD3.1.24 | TC3. 1.24 | Пример тестовых данных | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя выполнять резервное копирование тестового набора |
| RD3.1.25 | TC3. 1.25 | Пример тестовых данных | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя выполнять резервное копирование результатов прогона одиночного тестового случая |
| RD3.1.26 | TC3. 1.26 | Пример данных для резервного копирования | Тестовая конфигурация #1 | Проверка возможности восстановления зарегистрированным пользователем одиночного тестового случая |
| RD3.1.27 | TC3.1.27 | Пример данных для резервного копирования | Тестовая конфигурация #1 | Проверка возможности восстановления зарегистрированным пользователем тестового набора |
| RD3.1.28 | TC3. 1.28 | Пример данных для резервного копирования | Тестовая конфигурация #1 | Проверка возможности восстановления аутентифицированным пользователем результатов тестирования |

### *3.3. Дополнительные возможности*

Проекты тестов для дополнительных функциональных возможностей наподобие экспорта данных и функций выдачи справочной информации, а также многопользовательского режима, представлены в таблице 3.3.

**Таблица 3.3. Разработка теста для дополнительных свойств**

| Идентификатор требования | Идентификатор системного тестового случая | Входные данные теста | Тестовая конфигурация | Цель проведения теста |
|---|---|---|---|---|
| RD3.1.29 | TC3.1.29 | Пример тестовых данных | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя экспортировать одиночный тестовый случай |
| RD3.1.30 | TC3. 1.30 | Пример тестовых данных | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя экспортировать тестовый набор во внешнее приложение |
| RD3.1.31 | TC3. 1.31 | Пример тестовых данных | Тестовая конфигурация #1 | Проверка возможности зарегистрированного пользователя экспортировать результаты прогона тестов |
| RD3.1.32 | TC3. 1.32 | Нет | Тестовая конфигурация #1 | Проверка, что каждый экран имеет связанный с ним экран справочной информации, доступный для открытия |
| RD3.1.33 | TC3. 1.33 | Пример данных для проекта и теста | Тестовая конфигурация #1 | Проверка возможности одновременной работы 5 пользователей |

### *3.4. Установка продукта*

Каждая программная сборка, переданная команде тестировщиков, устанавливается в соответствии с процедурой установки, которую будет использовать заказчик. Однако для каждой сборки модули клиента и сервера устанавливаются только на подмножестве всех возможных комбинаций платформ и операционных систем, которые указаны в спецификации требований. Предполагается, что успешная установка на одной платформе UNIX создает прецедент для успешной установки для всех остальных UNIX-подобных платформ. То же справедливо и для платформ Windows. Этот подход утвержден у заказчика (см. сообщение электронной почты утверждающего лица из отдела маркетинга).

Для процесса установки какие-либо специальные тестовых случаи не создаются. Причина заключается в том, что непосредственно за документом по установке продукта, который передается заказчику, следует тестирование. После успешной установки продукта можно приступать к прогону тестов функциональных возможностей, перечисленных в таблице 3.1.

### *3.5. Тестирование графического интерфейса пользователя*

При тестировании графического интерфейса продукта TMT используется следующий подход:

- Графический интерфейс пользователя тестируется в браузерах Netscape Navigator и Microsoft Internet Explorer. При этом должен быть просмотрен полный состав интерфейса, а также протестированы возможности навигации в обоих браузерах.
- Все действия по тестированию выполняются в ручном режиме.

- Все дефекты отслеживаются и устраняются с помощью корпоративной системы отслеживания дефектов. Такой подход предполагает нахождение недоработок в графическом интерфейсе пользователя в ходе проведения различных оценок после завершения работы над проектом.

Какие-то специальное тесты для графического интерфейса пользователя не разрабатываются. Причина состоит в том, что применение интерфейса подразумевается во всех тестах свойств, а также в тестах резервного копирования/восстановления, которые описаны в табл. 3.1, 3.2 и 3.3. Если какой-либо из этих тестов завершается неудачно, то причина будет связана либо с графическим интерфейсом пользователя, либо с функциональными возможностями, которые доступны через этот интерфейс.

## 4. Информация о конфигурации тестов

Проекты тестов, перечисленные в таблицах 3.1, 3.2 и 3.3, связаны со специальными тестовыми конфигурациями. Диаграммы тестовых конфигураций # 1 и # 2 находятся в плане тестирования TMT, документ TMT-TP-08.

## 5. Тестовые случаи

В разделе представлены процедуры для всех тестов, приведенных таблицах 3.1, 3.2 и 3.3. Каждый тестовый случай должен выполняться в Netscape Navigator, а затем повторно в Microsoft Internet Explorer.

### *5.1. TC 3.1.1 Пользовательский интерфейс*

Сервер приложений должен иметь имя и IP-адрес. Для целей тестирования имени приложения и серверу присваивается псевдоним "TMT'.

**Случай 1**

Запустите Netscape Navigator. Введите URL-адрес "TMT" и нажмите клавишу "Enter".

***Ожидаемый результат:***
Отображается главное меню TMT.

### *5.2. TC 3.1.2 Навигация*

**Случай 1**

Запустите Netscape Navigator. Введите URL-адрес "TMT" и нажмите "Enter".

***Ожидаемый результат:***
Отображается главное меню TMT (Toolkit Main Menu).

**Случай 2**

Меню должно содержать "кнопки" или ссылки на другие страницы. Главное меню должно содержать следующие ссылки:

**Current Projects (Текущие проекты)**
**Completed Projects (Завершенные проекты)**
**Project Maintenance (Сопровождение проекта)**
**Test Case Maintenance (Сопровождение тестовых случаев)**
**Test Case Execution (Выполнение тестовых случаев)**
**Test Results (Результаты тестирования)**
**Utilities (Утилиты)**
**Help (Справка)**

***Ожидаемый результат:***
Меню имеет описанные выше метки.

**Случай 3**

Выберите в главном меню пункт "Current Projects"[1] ("Текущие проектыГУ

***Ожидаемый результат 1:***
Отображается экран, озаглавленный "Current Projects"("TeKym,He проекты"), который содержит проект или несколько проектов.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее на отсутствие текущих проектов (если данный тестовый случай выполняется до создания проектов).
В окне браузера щелкните на стрелке назад для возврата в главное меню.

***Ожидаемый результат 3:***
Пользователь должен вернуться в главное меню.

**Случай 4**

В главном меню выберите пункт "Completed Projects" ("Завершенные проектьП.

***Ожидаемый результат 1:***
Отображается экран с заголовком "Completed Projects"("3aBepLueHHbie проекты"), который содержит проект или несколько проектов.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее на отсутствие текущих проектов (если данный тестовый случай выполняется до создания проектов).
В окне браузера щелкните на стрелке назад для возврата в главное меню.

***Ожидаемый результат 3:***
Пользователь должен вернуться в главное меню.

**Случай 5**

В главном меню выберите ПУНКТ "Project Maintenance" ("Сопровождение проекта^.

***Ожидаемый результат 1:***
Отображается экран с заголовком "Project Ma1n1enapce"("Сопровождение проекта").

***Ожидаемый результат 2:***
Должны стать доступными следующие пункты:

**Create New Project (Создать новый проект)**
**Modify Project (Изменить проект)**
**Remove Project (Удалить проект)**
**Help (Справка)**

**Случай 6**

**В** меню Project Maintenance выберите ПУНКТ "Create New Project" ("Создать новый проектТ

***Ожидаемый результат 1:***
Появляется экран с запросом имени создаваемого проекта.
В окне браузера щелкните на стрелке назад для возврата в меню Project Maintenance.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Project Maintenance.

**Случай 7**

В меню Project Maintenance выберите ПУНКТ "Modify Project" ("Изменить проекте.

***Ожидаемый результат 1:***
На экране появляется запрос имени изменяемого проекта, а также выводится список доступных проектов.

***Ожидаемый результат 2:***
Появляется сообщение об ошибке, указывающее на отсутствие проектов, доступных для изменения (если данный тест выполняется до создания проектов).
В окне браузера щелкните на стрелке назад для возврата в меню Project Maintenance.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Project Maintenance.

**Случай 8:**

В меню Project Maintenance выберите пункт "Remove Project" ("Удалить проект").

***Ожидаемый результат 1:***
На экране появляется запрос имени удаляемого проекта, а также выводится список доступных проектов.

***Ожидаемый результат 2:***
Появляется сообщение об ошибке, указывающее на отсутствие проектов, доступных для удаления (если данный тест выполняется до создания проектов).
В окне браузера щелкните на стрелке назад для возврата в меню Project Maintenance.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Project Maintenance.

**Случай 9**

В меню Project Maintenance выберите пункт "Help" ("Справка").
Ожидаемый результат 1:
Отображается экран с подробным описанием всех пунктов меню Project Maintenance (в следующих версиях программы будет реализована контекстно-зависимая справочная система с индексами и возможностью поиска).
В окне браузера щелкните на стрелке назад для возврата в меню Project Maintenance.
Ожидаемый результат 2:
Пользователь должен вернуться в меню Project Maintenance.

**Случай 10**

В главном меню выберите ПУНКТ "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 2:***
Должны стать доступными следующие пункты:

**Create Test Case or Suite (Создать тестовый случай или набор)**
**Modify Test Case or Suite (Изменить тестовый случай или набор)**
**Remove Test Case or Suite (Удалить тестовый случай или набор)**
**Display Test (Показать тест)**
**Display Suite (Показать тестовый набор)**
**Help (Справка)**

**Случай 11**

В меню Test Case Maintenance выберите пункт "Create Test Case or Suite" ("Создать тестовый случай или набор").

***Ожидаемый результат 1:***
На экране появляется запрос имени проекта, для которого создается новый тестовый случай или набор. Пользователь может дважды щелкнуть на имени существующего проекта.

***Ожидаемый результат 2:***
Пользователь получает запрос на выбор между опциями Test (Тест) или Suite (Набор). Если выбрана опция Suite (Набор), поступает запрос об имени набора. Если выбрана опция Test (Тест), выдается запрос об имени тестового случая. В этот момент должен отобразиться экран, на котором будут вноситься данные, связанные с тестом.

**Случай 12**

В меню Test Case Maintenance выберите пункт "Modify Test Case or Suite" ("Изменить тестовый случай или набор").

***Ожидаемый результат 1:***
На экране появляется запрос об имени проекта, содержащего тестовый случай или набор, которые необходимо изменить. Пользователь может дважды щелкнуть на имени существующего проекта.

***Ожидаемый результат 2:***
Если проект идентифицирован, пользователь получает запрос на выбор между опциями Test (Тест) или Suite (Набор).
Если пользователь выбирает опцию Suite (Набор), поступает запрос на ввод имени тестового набора. Пользователь может дважды щелкнуть на имени существующего тестового набора.

***Ожидаемый результат 3:***
Отображается экран, указывающий, что доступные для изменения тестовые наборы отсутствуют (если этот тест выполняется до создания тестовых наборов).

***Ожидаемый результат 4:***
Если пользователь выбирает опцию Test (Тест), выдается запрос на ввод имени теста, который необходимо обновить. Пользователь может дважды щелкнуть на имени существующего проекта.

***Ожидаемый результат 5:***
Отображается экран, указывающий, что доступные для обновления тесты отсутствуют (если этот тест выполняется до создания тестовых случаев).
В окне браузера щелкните на стрелке назад для возврата в меню Test Case Maintenance.

***Ожидаемый результат 6:***
Пользователь должен вернуться в меню Test Case Maintenance.

**Случай 13**

В меню Test Case Maintenance выберите ПУНКТ "Remove Test Case or Suite" ("Удалить тестовый случай или набор").

***Ожидаемый результат 1:***
На экране появляется запрос об имени проекта, содержащего тестовый случай или набор, которые необходимо удалить. Пользователь может дважды щелкнуть на имени существующего проекта.

***Ожидаемый результат 2:***
Если проект идентифицирован, пользователь получает запрос на выбор между опциями Test (Тест) или Suite (Набор).
Если пользователь выбирает опцию Suite (Набор), поступает запрос на ввод имени тестового набора. Пользователь может дважды щелкнуть на имени существующего тестового набора.

***Ожидаемый результат 3:***
Отображается экран, указывающий, что доступные для удаления тестовые наборы отсутствуют (если этот тест выполняется до создания тестовых наборов).

***Ожидаемый результат 4:***
Если пользователь выбирает опцию Test (Тест), выдается запрос на ввод имени теста, который необходимо удалить. Пользователь может дважды щелкнуть на имени существующего проекта.

***Ожидаемый результат 5:***
Отображается экран, указывающий, что доступные для удаления тесты отсутствуют (если этот тест выполняется до создания тестовых случаев).
В окне браузера щелкните на стрелке назад для возврата в меню Test Case Maintenance.

***Ожидаемый результат 6:***
Пользователь должен вернуться в меню Test Case Maintenance.

**Случай 14**

В меню Test Case Maintenance выберите пункт "Help" ("Справка").

***Ожидаемый результат 1:***
Отображается экран с подробным описанием всех пунктов меню Test Case Maintenance (в следующих версиях программы будет реализована контекстно-зависимая справочная система с индексами и возможностью поиска).
В окне браузера щелкните на стрелке назад для возврата в меню Test Case Maintenance.

**Случай 15**

В меню Test Case Maintenance выберите пункт "Display Test" ("Показать тест").

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени теста, который необходимо отобразить (просмотреть). Ниже этого запроса находится список всех доступных тестов. Пользователь может дважды щелкнуть на требуемом тестовом случае.

***Ожидаемый результат 2:***
Появляется экран с сообщением о том, что доступные для просмотра тестовые случаи отсутствуют (если тест выполняется до создания тестовых случаев).

**Случай 16**

В меню Test Case Maintenance выберите пункт "Display Suite" ("Показать тестовый набор").

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени тестового набора, который необходимо отобразить (просмотреть). Ниже этого запроса находится список всех доступных тестов. Пользователь может дважды щелкнуть на требуемом тестовом наборе.

***Ожидаемый результат 2:***
Появляется экран с сообщением о том, что доступные для просмотра тестовые наборы отсутствуют (если тест выполняется до создания тестовых наборов).

**Случай 17**

В меню Test Case Maintenance выберите пункт "Help" ("Справка").

***Ожидаемый результат 1:***
Отображается экран с подробным описанием всех пунктов меню Test Case Maintenance (в следующих версиях программы будет реализована контекстно-зависимая справочная система с индексами и возможностью поиска).
В окне браузера щелкните на стрелке назад для возврата в меню Test Case Maintenance.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Test Case Maintenance.

**Случай 18**

В главном меню TMT выберите пункт "Test Case Execution" ("Выполнение тестовых случаев").

***Ожидаемый результат:***
Отображается экран с заголовком "Test Case Execution Menu" ("Меню выполнения тестовых случаев"). На экране должны присутствовать следующие пункты меню:

**Run Single Test (Прогнать одиночный тест)**
**Run Suite (Прогнать тестовый набор)**
**Create Run List (Создать список прогона)**
**Execute Run List (Выполнить список прогона)**
**Help (Справка)**

**Случай 19**

В меню Test Case Execution выберите ПУНКТ "Run Single Test" ("Прогнать одиночный тестТ

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени тестового случая, который требуется выполнить. Ниже этого запроса выводится список доступных тестовых случаев. Пользователь имеет возможность ввести имя прогоняемого тестового случая или дважды щелкнуть на существующем имени.

***Ожидаемый результат 2:***
Отображается экран с сообщением об отсутствии ранее созданных тестовых случаев (если тест выполняется до создания тестовых случаев).
Щелкните на стрелке назад в браузере.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Test Case Execution.

**Случай 20**

В меню Test Case Execution выберите ПУНКТ "Run Suite Case" ("Прогнать тестовый набор").

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени тестового набора, который необходимо выполнить. Ниже этого запроса выводится список доступных тестовых наборов. Пользователь имеет возможность ввести имя прогоняемого тестового набора или дважды щелкнуть на существующем имени.

***Ожидаемый результат 2:***
Отображается экран с сообщением об отсутствии созданных тестовых наборов (если тест выполняется до создания тестовых наборов).
Щелкните на стрелке назад в браузере.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Test Case Execution.

**Случай 21**

В меню Test Case Execution выберите ПУНКТ "Create Run List" ("Создать список прогонаТ

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени создаваемого списка прогона. Ниже этого запроса отображаются имена доступных списков прогона.
Щелкните на стрелке назад в браузере.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Test Case Execution.

**Случай 22**

В меню Test Case Execution выберите пункт "Execute Run List" ("Выполнить список прогона"У

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени списка прогона, который необходимо выполнить. Ниже этого запроса отображаются имена доступных списков прогона.
Щелкните на стрелке назад в браузере.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Test Case Execution.

**Случай 23**

В меню Test Case Execution выберите ПУНКТ "Help" ("Справка").

***Ожидаемый результат 1:***
Отображается экран с подробным описанием всех пунктов меню Test Case Maintenance (в следующих версиях программы будет реализована контекстно-зависимая справочная система с индексами и возможностью поиска).
В окне браузера щелкните на стрелке назад для возврата в меню Test Case Execution.
Снова щелкните на стрелке назад в браузере.

***Ожидаемый результат 2:***
Пользователь должен вернуться в главное меню Test Management Toolkit.

**Случай 24**

В главном меню Test Management Toolkit выберите ПУНКТ "Test Results" ("Результаты тестирования").

***Ожидаемые результаты:***
Пользователь должен получить доступ на экран с заголовком Test Results Menu (Меню результатов тестирования).

**Случай 25:**

В меню Test Results выберите ПУНКТ "Bug Summary" ("Сводный отчет по ошибкам").

***Ожидаемый результат 1:***
Это самое высокоуровневое визуальное представление, поэтому пользователю выдается запрос на ввод имени анализируемого проекта. Ниже этого запроса отображается список доступных проектов, как текущих, так и завершенных. Пользователь имеет возможность ввести имя проекта или же просто дважды щелкнуть на существующем имени.

Если пользователь выбирает прошлый или текущий проект, отображается экран, на котором выводится общее количество тестов, успешно и неудачно пройденных тестов, заблокированных тестов, общие затраты времени, а также дата прогона.

***Ожидаемый результат 2:***
Отображается экран с сообщением об отсутствии доступных результатов тестирования (если тест выполняется до выполнения или завершения проекта).
В окне браузера щелкните на стрелке назад для возврата в меню Test Results.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Test Results.

**Случай 26**

В меню Test Results выберите пункт "Single Test" ("Одиночный тест").

***Ожидаемый результат 1:***
Пользователь получает запрос на ввод имени тестового случая, который необходимо проанализировать. Ниже этого запроса выводится список всех доступных тестовых случаев, которые уже выполнены. Пользователь имеет возможность ввести имя теста или просто дважды щелкнуть на существующем имени.

***Ожидаемый результат 2:***
Отображается экран с сообщением об отсутствии доступных результатов тестирования (если тест выполняется до выполнения или завершения тестового случая).
В окне браузера щелкните на стрелке назад для возврата в меню Test Results.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Test Results.

**Случай 27**

В меню Test Results выберите пункт "Suite or Run List" ("Тестовый набор или список прогона"У

*Ожидаемый результат 1:*
Пользователь получает запрос на ввод имени тестового набора, который необходимо проанализировать. Ниже этого запроса выводится список всех доступных тестовых наборов, которые уже выполнены. Пользователь имеет возможность ввести имя тестового набора или просто дважды щелкнуть на существующем имени.

*Ожидаемый **результат** 2:*
Отображается экран с сообщением об отсутствии доступных результатов тестирования (если тест выполняется до выполнения или завершения тестового набора).
В окне браузера щелкните на стрелке назад для возврата в меню Test Results.

*Ожидаемый **результат** 3:*
Пользователь должен вернуться в меню Test Results.
В окне браузера щелкните на стрелке назад для возврата в главное меню Test Management Toolkit.

*Ожидаемый **результат** 4:*
Пользователь должен вернуться в главное меню Test Management Toolkit.

**Случай 28**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемые результаты:***
Пользователь получает доступ на экран с заголовком Utilities Main Menu (Меню утилит), в котором доступны следующие пункты:

**Create Trace Matrix (Создать матрицу прослеживаемости)**
**Backup Project (Резервное копирование проекта)**
**Backup Test Suite (Резервное копирование тестового набора)**
**Backup Test Case {Резервное копирование тестового случая)**
**Backup Test Results (Резервное копирование результатов прогона теста)**
**Restore Project (Восстановление проекта)**
**Restore Test Suite (Восстановление тестового набора)**
**Restore Test Case (Восстановление тестового случая)**
**Restore Test Results (Восстановление результатов прогона теста)**
**Export Project (Экспорт проекта)**
**Export Test Suite (Экспорт тестового набора)**
**Export Test Case (Экспорт тестового случая)**
**Export Test Results (Экспорт результатов прогона теста)**
**Help (Справка)**

**Случай 29**

В меню Utilities выберите пункт "Create Trace Matrix" ("Создать матрицу прослеживаемости").

*Ожидаемый **результат** 1:*
Пользователь получает запрос на ввод имени документа описания требований для его анализа. Ниже этого запроса отображается список всех доступных документов описания требований. Пользователь имеет возможность ввести имя или дважды щелкнуть на существующем имени. После выбора документов на экран выводится созданная матрица прослеживаемости, которую можно вывести на принтер или экспортировать в формат электронной таблицы.

*Ожидаемый **результат** 2:*
Пользователь получает сообщение об ошибке, указывающее, что каталог по умолчанию не содержит документов описания требований. Причина связана с тем, что документы либо не создавались, либо не находятся в нужном каталоге.
В окне браузера щелкните на стрелке назад для возврата в меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 30**

В меню Utilities выберите пункт "Backup Project" ("Резервное копирование проекта").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Project Backup (Резервное копирование проекта). После этого отображается запрос на ввод имени копируемого проекта со списком всех доступных проектов. Пользователь имеет возможность ввести имя проекта или дважды щелкнуть на существующем имени. После идентификации проекта отображается запрос на ввод местоположения резервной копии.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее, что доступные для резервного копирования проекты отсутствуют. Причиной может заключаться в том, что не было создано ни одного проекта.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 31**

В меню Utilities выберите пункт "Backup Test Suite" ("Резервное копирование тестового набора").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Suite Backup (Резервное копирование тестового набора). Отображается запрос на ввод имени копируемого тестового набора. Под этим запросом выводится список всех доступных тестовых наборов. Пользователь имеет возможность ввести имя тестового набора или дважды щелкнуть на существующем имени. После идентификации тестового набора отображается запрос на ввод местоположения резервной копии.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее, что доступные для резервного копирования тестовые наборы отсутствуют. Причиной может заключаться в том, что не было создано ни одного тестового набора.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 32**

В меню Utilities выберите пункт "Backup Test Case" ("Резервное копирование тестового случая").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Test Case Backup (Резервное копирование тестового случая). Отображается запрос на ввод имени копируемого тестового случая. Под этим запросом выводится список всех доступных тестовых случаев. Пользователь имеет возможность ввести имя тестового случая или дважды щелкнуть на существующем имени. После идентификации тестового случая отображается запрос на ввод местоположения резервной копии.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее, что доступные для резервного копирования тестовые случаи отсутствуют. Причиной может заключаться в том, что не было создано ни одного тестового случая.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 33**

В меню Utilities выберите ПУНКТ "Backup Test Results" ("Резервное копирование результатов прогона теста").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Test Results Backup (Резервное копирование результатов прогона теста). Отображается запрос на ввод имени копируемых результатов прогона теста. Под этим запросом выводится список всех доступных результатов прогона тестов. Пользователь имеет возможность ввести имя результатов прогона теста или дважды щелкнуть на существующем имени. После идентификации результатов прогона теста отображается запрос на ввод местоположения резервной копии.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее, что доступные для резервного копирования результаты прогона тестов отсутствуют. Причиной может заключаться в том, что не было получено ни одного результата прогона тестов.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 34:**

В меню Utilities выберите ПУНКТ "Restore Project" ("Восстановление проекта").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Restore Project (Восстановление проекта). Выдается запрос на ввод имени резервной копии для восстановления. После идентификации резервной копии пользователь получает запрос на ввод местоположения, куда требуется восстановить резервную копию. Поддерживается информация по умолчанию.
В окне браузера щелкните на стрелке назад для возврата в меню Utilities.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Utilities.

**Случай 35**

В меню Utilities выберите ПУНКТ "Restore Test Suite" ("Восстановление тестового набора").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Restore Test Suite (Восстановление тестового набора). Выдается запрос на ввод имени резервной копии для восстановления. После идентификации резервной копии пользователь получает запрос на ввод местоположения, куда требуется восстановить резервную копию. Поддерживается информация по умолчанию.
В окне браузера щелкните на стрелке назад для возврата в меню Utilities.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Utilities.

**Случай 36**

В меню Utilities выберите пункт "Restore Test Case" ("Восстановление тестового случая").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Restore Test Case (Восстановление тестового случая). Выдается запрос на ввод имени резервной копии для восстановления. После идентификации резервной копии пользователь получает запрос на ввод местоположения, куда требуется восстановить резервную копию. Поддерживается информация по умолчанию.
В окне браузера щелкните на стрелке назад для возврата в меню Utilities.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Utilities.

**Случай 37**

В главном меню Utilities выберите пункт "Restore Test Results" ("Восстановление результатов прогона теста").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Restore Test Results (Восстановление результатов прогона теста). Выдается запрос на ввод имени резервной копии для восстановления. После идентификации резервной копии пользователь получает запрос на ввод местоположения, куда требуется восстановить резервную копию. Поддерживается информация по умолчанию.
В окне браузера щелкните на стрелке назад для возврата в меню Utilities.

***Ожидаемый результат 2:***
Пользователь должен вернуться в меню Utilities.

**Случай 38**

В меню Utilities выберите пункт Export Project (Экспорт проекта).

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Export Project (Экспорт проекта). Отображается запрос на ввод имени проекта, который необходимо экспортировать. Ниже этого запроса выводится список всех доступных проектов. Пользователь имеет возможность ввести имя проекта или дважды щелкнуть на существующем имени. После идентификации проекта пользователь получает запрос о том, куда следует поместить экспортированный проект.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее на отсутствие проектов, доступных для экспорта. Причина может заключаться в том, что ни один проект еще не создавался.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 39**

В меню Utilities выберите пункт "Export Test Suite" ("Экспорт тестового набора").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Export Test Suite (Экспорт тестового набора). Отображается запрос на ввод имени тестового набора, который необходимо экспортировать. Ниже этого запроса выводится список всех доступных тестовых наборов. Пользователь имеет возможность ввести имя тестового набора или дважды щелкнуть на существующем имени. После идентификации тестового набора пользователь получает запрос о том, куда следует поместить экспортированный тестовый набор.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее на отсутствие тестовых наборов, доступных для экспорта. Причина может заключаться в том, что ни один тестовый набор еще не создавался.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 40**

В меню Utilities выберите пункт "Export Test Case" ("ЭКСПОРТ тестового случая").

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Export Test Case (Экспорт тестового случая). Отображается запрос на ввод имени тестового случая, который необходимо экспортировать. Ниже этого запроса выводится список всех доступных тестовых случаев. Пользователь имеет возможность ввести имя тестового случая или дважды щелкнуть на существующем имени. После идентификации тестового случая пользователь получает запрос о том, куда следует поместить экспортированный тестовый случай.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее на отсутствие тестовых случаев, доступных для экспорта. Причина может заключаться в том, что ни один тестовый случай еще не создавался.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 41**

В меню Utilities выберите пункт "Export Test Results" ("ЭКСПОРТ результатов прогона тестаТ

***Ожидаемый результат 1:***
Пользователь получает доступ на экран с заголовком Export Test Results (Экспорт результатов прогона теста). Отображается запрос на ввод имени результатов прогона теста, которые необходимо экспортировать. Ниже этого запроса выводится список всех доступных результатов прогона теста. Пользователь имеет возможность ввести имя результатов прогона теста или дважды щелкнуть на существующем имени. После идентификации результатов прогона теста пользователь получает запрос о том, куда следует поместить их экспортированную версию.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке, указывающее на отсутствие результатов прогона теста, доступных для экспорта. Причина может заключаться в том, что ни один результат прогона теста еще не был получен.
В окне браузера щелкните на стрелке назад для возврата в главное меню Utilities.

***Ожидаемый результат 3:***
Пользователь должен вернуться в меню Utilities.

**Случай 42**

В главном меню Utilities выберите опцию Help.

***Ожидаемый результат 1:***
Отображается экран с подробным описанием всех пунктов меню Utilities (в следующих версиях программы будет реализована контекстно-зависимая справочная система с индексами и возможностью поиска).
В окне браузера щелкните на стрелке назад для возврата в меню Utilities.
Снова щелкните в окне броузера на стрелке назад для возврата в главное меню Test Management Toolkit.

***Ожидаемый результат 2:***
Пользователь должен вернуться в главное меню Test Management Toolkit.

### 5.3. *ТС 3.1.3 Аутентификация пользователей — клиент*

Для проведения этого теста создаются следующие учетные записи:

| Идентификатор пользователя | Пароль |
|---|---|
| Admin | Admin |
| Userl | password |
| User2 | password |
| User3 | password |
| User4 | password |
| User5 | password |

**Случай 1**

После набора в адресной строке браузера TMT и нажатия клавиши Enter пользователь должен получить запрос на ввод имени и пароля. Инициируйте процесс регистрации, набрав в строке псевдоним TMT и нажав клавишу Enter.

***Ожидаемый результат 1:***
В центре экрана открывается окно с запросом User Name (Имя пользователя) и Password (Пароль).
Введите "User9" и нажмите клавишу Enter, не набирая пароль.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "User9" для имени пользователя, а в качестве пароля введите "password".

***Ожидаемый результат 3:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "Userl" для имени пользователя, а в качестве пароля укажите "pickle".

***Ожидаемый результат 4:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "Userl" для имени пользователя, а в качестве пароля укажите "password".

***Ожидаемый результат 5:***
Отображается главное меню Test Management Toolkit.

## *5.4. TC 3.1.4 Аутентификация пользователей —администратор*

**Клиентская сторона**

**Случай 1**

После набора в адресной строке браузера TMT и нажатия клавиши Enter пользователь должен получить запрос на ввод имени и пароля. Инициируйте процесс регистрации, набрав в строке псевдоним TMT и нажав клавишу Enter.

***Ожидаемый результат 1:***
В центре экрана открывается окно с запросом User Name (Имя пользователя) и Password (Пароль).
Введите "User9" и нажмите клавишу Enter, не набирая пароль.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "User9" для имени пользователя, а в качестве пароля укажите "password".

***Ожидаемый результат 3:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").

Введите "User1" для имени пользователя, а в качестве пароля укажите "pickle".

***Ожидаемый результат 4:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "Admin" для имени пользователя, а в качестве пароля укажите "Admin".

***Ожидаемый результат 5:***
Отображается главное меню Test Management Toolkit.

**Серверная сторона**

**Случай 2**

После набора в адресной строке браузера TMTADMIN и нажатия клавиши Enter пользователь должен получить запрос на ввод имени и пароля. Инициируйте процесс регистрации, набрав в строке псевдоним TMTADMIN и нажав клавишу Enter.

***Ожидаемый результат 1:***
В центре экрана открывается окно с запросом User Name (Имя пользователя) и Password (Пароль).
Введите "User9" и нажмите клавишу Enter, не набирая пароль.

***Ожидаемый результат 2:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "User9" для имени пользователя, а в качестве пароля укажите "password".

***Ожидаемый результат 3:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "User1" для имени пользователя, а в качестве пароля укажите "pickle".

***Ожидаемый результат 4:***
Отображается сообщение об ошибке "You entered an invalid Username or Password" ("Введено неверное имя пользователя или пароль").
Введите "Admin" для имени пользователя, а в качестве пароля укажите "Admin".

***Ожидаемый результат 5:***
Отображается главное меню Test Management Toolkit.

### *5.5. TC 3.1.5 Текущие проекты*

После регистрации в системе пользователь имеет возможность просматривать текущие проекты. Для этого необходимо из главного меню Test Management Toolkit выбрать пункт "Current Projects" ("Текущие проекты").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Current Projects" ("Текущие проекты").

***Ожидаемый результат:***
Отображается экран с заголовком Current Projects (Текущие проекты). С именами проектов связано общее количество тестов, количество пройденных тестов, а также количество сбойных, заблокированных и прошедших тестов. Если тесты уже прогнаны и соответствующие данные собраны, для проектов отображается также время, оставшееся до конца тестирования, и общее время выполнения.

## 5.6. *TC 3.1.6 Завершенные проекты*

После регистрации в системе пользователь имеет возможность просматривать текущие проекты. Для этого необходимо из главного меню Test Management Toolkit выбрать пункт "Completed Projects" ("Завершенные проекты").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Completed Projects" ("Завершенные проекты").

***Ожидаемый результат:***
Отображается экран с заголовком Completed Projects (Завершенные проекты). С именами проектов связано общее количество тестов, количество пройденных тестов, а также количество сбойных, заблокированных и прошедших тестов. Отображается также и общее время в часах и минутах.

### 5.7. *TC 3.1.7 Создание проекта*

После регистрации в системе пользователь имеет возможность создавать новые проекты. Для этого необходимо в меню Project Maintenance выбрать пункт "Create New Project" ("Создать новый проект").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Project Maintenance" ("Сопровождение проекта").

***Ожидаемый результат 1:***
Отображается экран с пунктами меню Project Maintenance (Сопровождение проекта).
В меню Project Maintenance (Сопровождение проекта) выберите пункт "Create New Project" ("Создать новый проект").

***Ожидаемый результат 2:***
Пользователю выдается запрос на ввод имени нового проекта. После ввода имени нового проекта отображается запрос о добавлении индивидуальных тестовых случаев или наборов.
Пользователь имеет возможность просматривать список доступных тестовых случаев и наборов.
Пользователь имеет возможность выбирать любой тестовый случай или набор, а также выбирать все тестовые случаи или наборы.
По завершении выбора появляется возможность сохранить проект (Save) или отменить действия (Cancel).
Независимо от выбранной опции пользователь возвращается в меню Project Maintenance.

### 5.8. *TC 3.1.8 Изменение проекта*

После регистрации в системе пользователь имеет возможность изменять проекты. Для этого необходимо в меню Project Maintenance выбрать пункт "Modify Project" ("Изменить проект").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Project Maintenance" ("Сопровождение проекта").

***Ожидаемый результат 1:***
Отображается экран с пунктами меню Project Maintenance (Сопровождение проекта).
В меню Project Maintenance выберите пункт "Modify Project" ("Изменить проект").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени проекта, который требуется изменить.
Пользователь также получает список всех доступных проектов.
Пользователь имеет возможность водить имя изменяемого проекта или просто дважды щелкать на существующем имени.
После внесения изменений в проект появляется возможность сохранить проект (Save) или отменить действие (Cancel).
Независимо от выбранной опции пользователь возвращается в меню Project Maintenance.

### *5.9. TC 3.1.9 Удаление проекта*

После регистрации в системе пользователь имеет возможность удалять проекты. Для этого необходимо в меню Project Maintenance выбрать пункт "Remove Project" ("Удалить проект").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Project Maintenance" ("Сопровождение проекта").

***Ожидаемый результат 1:***
Отображается экран с пунктами меню Project Maintenance (Сопровождение проекта).
В меню Project Maintenance выберите пункт "Remove Project" ("Удалить проект").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени проекта, который требуется удалить.
Пользователь также получает список всех доступных проектов.
Пользователь имеет возможность водить имя удаляемого проекта или просто дважды щелкать на существующем имени.
Далее пользователь возвращается в меню Project Maintenance.

### *5.10. TC 3.1.10 Создать тестовый случай или набор*

После регистрации в системе пользователь имеет возможность создавать тестовые случаи. Для этого необходимо в меню Test Case Maintenance выбрать пункт "Create Test Case or Suite" ("Создать тестовый случай или набор").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Maintenance (Сопровождение тестовых случаев).
Выберите пункт "Create Test Case or Suite" ("Создать тестовый случай или набор").

***Ожидаемый результат 2:***
Пользователь получает запрос на выбор между созданием тестового случая (Test Case) или тестового набора (Test Suite).
После того, как пользователь определился с тем, что создает, выдается запрос на ввод имени теста.
После ввода имени тестового случая или набора появляется экран ввода данных.

***Ожидаемый результат 3:***
Отображается форма для ввода данных.
После ввода всей необходимой информации, связанной с тестом, пользователю предоставляется возможность сохранить тест (Save) или отменить действие (Cancel).
Независимо от выбранной опции, пользователь возвращается в меню Test Case Maintenance.

### *5.11. TC 3.1.11 Изменить тестовый случай или набор*

После регистрации в системе пользователь получает возможность изменять тестовые случаи. Для этого необходимо в меню Test Case Maintenance выбрать пункт "Modify Test Case or Suite" ("Изменить тестовый случай или набор").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Maintenance (Сопровождение тестовых случаев).
Выберите пункт "Modify Test Case or Suite" ("Изменить тестовый случай или набор").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая или набора, который требуется изменить. Отображается список всех доступных тестовых случаев и наборов. Пользователь имеет возможность ввести имя тестового случая или набора либо дважды щелкнуть на существующем имени.

***Ожидаемый результат 3:***
После выполнения всех необходимых действий пользователю предоставляется возможность сохранить изменения (Save) или отменить действия (Cancel).
Независимо от выбранной опции, пользователь возвращается в меню Test Case Maintenance.

### *5.12. TC 3.1.12 Удалить тестовый случай или набор*

После регистрации в системе пользователь получает возможность удалять тестовые случаи. Для этого необходимо в меню Test Case Maintenance выбрать пункт "Remove Test Case or Suite" ("Удалить тестовый случай или набор").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Maintenance (Сопровождение тестовых случаев).
Выберите пункт "Remove Test Case or Suite" ("Удалить тестовый случай или набор").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая или набора, который требуется удалить. Отображается список всех доступных тестовых случаев и наборов. Пользователь имеет возможность ввести имя тестового случая или набора либо дважды щелкнуть на существующем имени для удаления.

***Ожидаемый результат 3:***
Независимо от выбранной опции, пользователь возвращается в меню Test Case Maintenance.

## 5.13. *TC 3.1.13 Показать тест*

После регистрации в системе пользователь получает возможность отображать (просматривать) тестовые случаи. Для этого необходимо в меню Test Case Maintenance выбрать пункт "Display Test Case or Suite" ("Показать тестовый случай или набор").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Maintenance (Сопровождение тестовых случаев).
Выберите пункт "Display Test Case or Suite" ("Показать тестовый случай или набор").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая или набора, который требуется отобразить. Выводится список всех доступных тестовых случаев и наборов. Пользователь имеет возможность ввести имя тестового случая или набора либо дважды щелкнуть на существующем имени.
Тестовый случай отображается в режиме только для чтения в том же формате, в котором он вводился.

По завершении работы пользователю возвращается в меню Test Case Maintenance.

### *5.14. TC 3.1.14 Показать тестовый набор*

После регистрации в системе пользователь получает возможность отображать (просматривать) тестовые наборы. Для этого необходимо в меню Test Case Maintenance выбрать пункт "Display Test Case or Suite" ("Показать тестовый случай или набор").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Case Maintenance" ("Сопровождение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Maintenance (Сопровождение тестовых случаев).
Выберите пункт "Display Test Case or Suite" ("Показать тестовый случай или набор").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового набора, который требуется отобразить. Выводится список всех доступных тестовых наборов. Пользователь имеет возможность ввести имя тестового набора либо дважды щелкнуть на существующем имени.

Не экране отображаются тестовые случаи, которые входят в набор. Пользователь имеет возможность по двойному щелчку на тестовом случае просматривать связанную с ним информацию.
По завершении работы пользователю возвращается в меню Test Case Maintenance.

### *5.15. TC 3.1.15 Прогнать одиночный тест*

После регистрации в системе пользователь получает возможность прогонять тестовые случаи. Для этого необходимо в меню Test Case Execution (Выполнение тестовых случаев) выбрать пункт "Run Single Test" ("Прогнать одиночный тест").

**Случай 1**

В главном меню Test Management Toolkit выберите опцию "Test Case Execution" ("Выполнение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Execution (Выполнение тестовых случаев).
Выберите пункт "Run Single Test" ("Прогнать одиночный тест").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая, который необходимо прогнать. Отображается список всех доступных тестовых случаев. Пользователь имеет возможность ввести имя тестового случая или набора или дважды щелкнуть на существующем имени.

***Ожидаемый результат 3:***
Отображаются детальные шаги выбранного тестового случая. Пользователь выполняет указанные задания, описанные в тестовом случае. Далее пользователь должен выбрать один из следующих вариантов:

- Тест прошел, тест не прошел или тест заблокирован и прогнан быть не может.
- После выбора пользователем одного из вариантов выполняется возврат в меню Test Case Execution.

### *5.16. TC 3.1.15 Выполнение набора тестов*

После регистрации в системе пользователь получает возможность прогонять тестовые наборы. Для этого необходимо в меню Test Execution выбрать пункт "Run Suite" ("Прогнать тестовый набор").

**Случай 1**

В главном меню Test Management Toolkit выберите опцию "Test Case Execution" ("Выполнение тестовых случаев").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Case Execution (Выполнение тестовых случаев).
Выберите пункт "Run Suite" ("Прогнать тестовый набор").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового набора, который необходимо прогнать. Отображается список всех доступных тестовых наборов. Пользователь имеет возможность ввести имя тестового набора или набора или дважды щелкнуть на существующем имени.

***Ожидаемый результат 3:***
Отображаются детальные шаги тестовых случаев, входящих в выбранный набор. Пользователь выполняет указанные задания, описанные в каждом тестовом случае. Далее пользователь должен выбрать один из следующих вариантов для каждого теста:

- Тест прошел, тест не прошел или тест заблокирован и прогнан быть не может.
- Результаты выполнения тестового набора носят накопительный характер. Необходимо выполнить все тесты из выбранного тестового набора. Если прогон одного из тестов оказывается неудачным, таким же признается и прогон всего тестового набора.

После выбора пользователем одного из вариантов выполняется возврат в меню Test Case Execution.

### *5.17. TC 3.1.17 Сводный отчет по ошибкам*

Это представление состояния действий, связанных с тестированием, на уровне проектов. После регистрации в системе пользователь получает возможность просматривать сводный отчет по ошибкам (Bug Summary). Для этого необходимо в главном меню Test Management Toolkit выбрать пункт "Test Results" ("Результаты тестирования").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Results" ("Результаты тестирования").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Results (Результаты тестирования).
В меню Test Results (Результаты тестирования) выберите пункт "Bug Summary" ("Сводный отчет по ошибкам ").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени проекта, который необходимо проанализировать. Отображается список всех текущих и завершенных проектов. Пользователь имеет возможность ввести имя проекта или дважды щелкнуть на существующем имени.
Вместе с именем проекта отображается общее количество пройденных тестов, тестов, которые не прошли, и тестов, оказавшихся заблокированными. Каждое число отображается в форме процентного отношения к общему количеству тестов.
По завершении выполняется возврат в меню Test Results.

### *5.18. TC 3.1.18 Результаты тестирования/Одиночный тест*

Это представление состояния действий, связанных с тестированием, на уровне тестовых случаев. После регистрации в системе пользователь получает возможность просматривать результаты тестирования для одиночных тестов. Для этого необходимо в главном меню Test Management Toolkit выбрать пункт "Test Results" ("Результаты тестирования").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Results" ("Результаты тестирования").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Results (Результаты тестирования).
В меню Test Results (Результаты тестирования) выберите пункт "Single Test" ("Одиночный тест").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая, который необходимо проанализировать. Отображается список всех доступных результатов прогона тестовых случаев. Пользователь имеет возможность ввести имя тестового случая или дважды щелкнуть на существующем имени.
Вместе с именем тестового случая отображаются результат его прогона в виде "прошел'Тне прошелТзаблокирован".
По завершении выполняется возврат в меню Test Results.

### *5.19 TC 3.1.19 Результаты тестирования/Тестовый набор*

Это представление состояния действий, связанных с тестированием, на уровне тестовых наборов. После регистрации в системе пользователь получает возможность просматривать результаты тестирования для тестовых наборов и списков прогона. Для этого необходимо в главном меню Test Management Toolkit выбрать пункт "Test Results" ("Результаты тестирования").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Test Results" ("Результаты тестирования").

***Ожидаемый результат 1:***
Отображается экран с заголовком Test Results (Результаты тестирования).
В меню Test Results (Результаты тестирования) выберите пункт "Suite or Run List" ("Тестовый набор или список прогона").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового набора или списка прогона, которые необходимо проанализировать. Отображаются имена всех доступных результатов выполнения тестовых наборов и списков прогона. Пользователь имеет возможность ввести имя тестового набора или списка прогона либо дважды щелкнуть на существующем имени.

Вместе с именем тестового набора или списка прогона отображаются результат его прогона в виде "прошел'Т'не прошел'Тзаблокирован".

По завершении выполняется возврат в меню Test Results.

### 5.20. *TC 3.1.20 Создать матрицу прослеживаемости*

После регистрации в системе пользователь получает возможность создать матрицу прослеживаемости требований. Для этого необходимо в меню Utilities (Утилиты) выбрать пункт "Create Trace Matrix" ("Создать матрицу прослеживаемости").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран с заголовком Utilities Menu (Меню утилит).
В меню Utilities (Утилиты) выберите пункт "Create Trace Matrix" ("Создать матрицу прослеживаемости").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени документа описания требований, который должен анализироваться.
Размещение этого документа жестко определено. На том же экране, где содержится запрос пользователя на ввод имени этого документа, указывается список всех доступных документов описания требований. Пользователь имеет возможность вводить имя необходимого документа или дважды щелкать на имени соответствующего файла.

***Ожидаемый результат 3:***
После выбора пользователем файла открывается новое окно, содержащее в одном столбце требования к продукту/проекту. Здесь будут присутствовать столбцы с соответствующими тестовыми слу-

чаями, доказывающими то или иное требование. Данная форма предназначена только для чтения и может быть выведена на печать для ручного заполнения. Пользователь имеет возможность выводить форму на принтер, а также экспортировать ее в формате электронной таблицы.
По завершении выполняется возврат в меню Utilities.

### *5.21. TC 3.1.21 Резервное копирование/Тестовые случаи*

После регистрации в системе пользователь получает возможность выполнить резервное копирование отдельных тестовых случаев. Для этого необходимо в меню Utilities выбрать пункт "Backup Test Case" ("Резервное копирование тестового случая").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран с заголовком Utilities Menu (Меню утилит).
В меню Utilities выберите пункт "Backup Test Case" ("Резервное копирование тестового случая").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая, который должен копироваться. Отображается список всех доступных тестовых случаев. Пользователь имеет возможность вводить имя тестового случая или дважды щелкнуть на существующем имени.

***Ожидаемый результат 3:***
После выбора пользователем тестового случая для копирования выдается запрос на ввод конечного расположения резервной копии. Расположение резервной копии варьируется в зависимости от системы, однако им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW. Кроме того, имеется возможность изменить имя резервной копии, которым по умолчанию является имя тестового случая.

По завершении выполняется возврат в меню Utilities.

### *5.22. TC 3.1.22 Резервное копирование/Тестовые наборы*

После регистрации в системе пользователь получает возможность выполнить резервное копирование тестовых наборов. Для этого необходимо в меню Utilities выбрать пункт "Backup Test Suite" ("Резервное копирование тестового набора").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите пункт "Backup Test Suite" ("Резервное копирование тестового набора").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового набора, который должен копироваться. Отображается список всех доступных тестовых наборов. Пользователь имеет возможность вводить имя тестового набора или дважды щелкнуть на существующем имени.

***Ожидаемый результат 3:***
После выбора пользователем тестового набора для копирования выдается запрос на ввод конечного расположения резервной копии. Расположение резервной копии варьируется в зависимости от системы, однако им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW. Кроме того, имеется возможность изменить имя резервной копии, которым по умолчанию является имя тестового набора.

По завершении выполняется возврат в меню Utilities.

### *5.23. TC 3.1.23 Резервное копирование/Результаты прогона тестов*

После регистрации в системе пользователь получает возможность выполнить резервное копирование результатов прогона тестов. Для этого необходимо в меню Utilities выбрать пункт "Backup Test Results" ("Резервное копирование результатов прогона теста").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***

Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите пункт "Backup Test Results" ("Резервное копирование результатов прогона теста").

***Ожидаемый результат 2:***

Пользователь получает запрос на ввод имени результатов прогона теста, которые должен копироваться. Отображается список всех доступных результатов прогона тестов. Пользователь имеет возможность вводить имя результатов прогона теста или дважды щелкнуть на существующем имени.

***Ожидаемый результат 3:***

После выбора пользователем результатов прогона теста для копирования выдается запрос на ввод конечного расположения резервной копии. Расположение резервной копии варьируется в зависимости от системы, однако им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW. Кроме того, имеется возможность изменить имя резервной копии, которым по умолчанию является имя результатов прогона теста.

По завершении выполняется возврат в меню Utilities.

### *5.24. TC 3.1.24 Восстановление/Тестовые случаи*

После регистрации в системе пользователь получает возможность восстанавливать тестовые случаи по оригинальному их расположению. Для этого необходимо в меню Utilities выбрать пункт "Restore Test Case" ("Восстановление тестового случая").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***

Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите пункт "Restore Test Case" ("Восстановление тестового случая").

***Ожидаемый результат 2:***

Пользователь получает запрос о месте расположения резервной копии, КОТОРУЮ необходимо восстановить.
Расположение резервной копии варьируется в зависимости от системы, однако им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW. После указания расположения пользователь получает запрос на ввод имени сохраненного тестового случая, который необходимо восстановить. Отображается список всех доступных архивов тестовых случаев. Пользователь может либо вручную ввести имя тестового случая, либо дважды щелкнуть на имени файла с резервной копией.

После выбора пользователем места расположения и файла, утилита восстанавливает тестовый случай по месту его оригинального расположения.

По завершении выполняется возврат в меню Utilities.

### 5.25. *TC 3.1.25 Восстановление/Тестовые наборы*

После регистрации в системе пользователь получает возможность восстанавливать тестовые наборы по оригинальному их расположению. Для этого необходимо в меню Utilities выбрать пункт "Restore Test Suite" ("Восстановление тестового набора").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите пункт "Restore Test Suite" ("Восстановление тестового набора").

***Ожидаемый результат 2:***
Пользователь получает запрос о месте расположения резервной копии, которую необходимо восстановить.
Расположение резервной копии варьируется в зависимости от системы, однако им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW. После указания расположения пользователь получает запрос на ввод имени сохраненного тестового набора, который необходимо восстановить. Отображается список всех доступных архивов тестовых наборов. Пользователь может либо вручную ввести имя тестового набора, либо дважды щелкнуть на имени файла с резервной копией.

После выбора пользователем места расположения и файла, утилита восстанавливает тестовый набор по месту его оригинального расположения.

По завершении выполняется возврат в меню Utilities.

### 5.26. *TC 3.1.26 Восстановление/Результаты прогона тестов*

После регистрации в системе пользователь получает возможность восстанавливать результаты тестирования по оригинальному их расположению. Для этого необходимо в меню Utilities выбрать пункт "Restore Test Results" ("Восстановление результатов прогона теста").

**Случай 1**

В главном меню Test Management Toolkit выберите ПУНКТ "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите ПУНКТ "Restore Test Results" ("Восстановление результатов прогона теста").

***Ожидаемый результат 2:***
Пользователь получает запрос о месте расположения резервной копии, КОТОРУЮ необходимо восстановить.
Расположение резервной копии варьируется в зависимости от системы, однако им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW. После указания расположения пользователь получает запрос на ввод имени сохраненных результатов прогона теста, которые необходимо восстановить. Отображается список всех доступных архивов результатов прогона теста. Пользователь может либо вручную ввести имя результатов прогона теста, либо дважды щелкнуть на имени файла с резервной копией.

После выбора пользователем места расположения и файла, утилита восстанавливает результаты прогона теста по месту их оригинального расположения.

По завершении выполняется возврат в меню Utilities.

### *5.27. TC 3.1.27 Экспорт/Тестовые случаи*

После регистрации в системе пользователь имеет возможность выполнять экспорт тестовых случаев для использования их в других приложениях. Для этого необходимо в меню Utilities выбрать пункт "Export Test Case" ("Экспорт тестового случая").

**Случай 1**

В главном меню Test Management Toolkit выберите ПУНКТ "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите пункт "Export Test Case" ("ЭКСПОРТ тестового случая").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового случая, который необходимо экспортировать.
Ниже отображаются все доступные тестовые случаи. Пользователь имеет возможность либо ввести вручную имя экспортируемого тестового случая, либо дважды щелкнуть на имени в списке.
После ввода имени тестового случая, который необходимо экспортировать, выдается запрос на выбор целевого устройства. В зависимости от конкретной системы, им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW.

После указания имени тестового случая и целевого расположения необходимо щелкнуть на кнопке "Continue" ("Продолжить"), в результате чего выполняется экспорт.

По завершении выполняется возврат в меню Utilities.

### *5.28. TC 3.1.28 Экспорт/Тестовые наборы*

После регистрации в системе пользователь имеет возможность экспортировать тестовые наборы для использования в других приложениях. Для этого необходимо в меню Utilities выбрать пункт "Export Test Suite" ("Экспорт тестового набора").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты").

***Ожидаемый результат 1:***
Отображается экран под заголовком Utilities Menu.
В меню Utilities выберите пункт "Export Test Suite" ("Экспорт тестового набора").

***Ожидаемый результат 2:***
Пользователь получает запрос на ввод имени тестового набора, который необходимо экспортировать.
Ниже отображаются все доступные тестовые наборы. Пользователь имеет возможность либо ввести вручную имя экспортируемого тестового набора, либо дважды щелкнуть на имени в списке.
После ввода имени тестового набора, который необходимо экспортировать, выдается запрос на выбор целевого устройства. В зависимости от конкретной системы, им может быть гибкий диск, локальный жесткий диск, сетевой диск или магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW.

После указания имени тестового набора и целевого расположения необходимо щелкнуть на кнопке "Continue" ("Продолжить"), в результате чего выполняется экспорт.

По завершении выполняется возврат в меню Utilities.

### *5.29. TC 3.1.29 Экспорт/Результаты прогона тестов*

После регистрации в системе пользователь имеет возможность экспортировать результаты прогона тестов для использования в других приложениях. Для этого необходимо в меню Utilities выбрать пункт "Export Test Results" ("Экспорт результатов прогона теста").

**Случай 1**

В главном меню Test Management Toolkit выберите пункт "Utilities" ("Утилиты!.

***Ожидаемый результат 1:***

Отображается экран под заголовком Utilities Menu.

В меню Utilities выберите пункт "Export Test Results" ("Экспорт результатов прогона теста"У

***Ожидаемый результат 2:***

Пользователь получает запрос на ввод имени результатов прогона теста, которые необходимо экспортировать.

Ниже отображаются все доступные результаты прогона тестов. Пользователь имеет возможность либо ввести вручную имя экспортируемых результатов прогона теста, либо дважды щелкнуть на имени в списке.

После ввода имени результатов прогона теста, которые необходимо экспортировать, выдается запрос на выбор целевого устройства. В зависимости от конкретной системы, им может быть гибкий диск, локальный жесткий диск, сетевой диск или-магнитная лента. В качестве вариантов также можно рассматривать дисководы CDR или CDRW.

После указания имени результатов прогона теста и целевого расположения необходимо щелкнуть на кнопке "Continue" ("Продолжить"), в результате чего выполняется экспорт.

По завершении выполняется возврат в меню Utilities.

### *5.30. TC 3.1.30 Справка*

С каждым экраном связан собственный уникальный экран справочной информации. Справочный экран поддерживает подробную информацию по всем возможностям, доступным пользователю в каждом меню. В справочную информацию входят примеры, синтаксис, предупреждения и другая полезная информация. В будущих версиях планируется реализовать интегрированную справочную подсистему, которая станет глобальной для всего приложения. В систему будет входить контекстно-зависимая справка, индексы и поисковый механизм. В настоящий момент доступна только жестко закодированная справочная информация.

**Случай 1**

В любом меню выберите пункт Help (СправкаУ

***Ожидаемый результат:***

Экран содержит записи для каждого пункта меню. Синтаксис и семантика фраз корректны.

### *5.31. TC 3.1.31 Многопользовательские функциональные возможности*

В данном случае предпринимается попытка синхронизировать действия с целью обеспечения максимальной нагрузки на определенные компоненты. Имеется тщательно скорректированная последовательность действий, которая вызовет повышенную нагрузку на определенные компоненты. Датой тестирования должно быть 8 октября. Для минимизации влияния со стороны сетевой среды тестирование должно начаться не ранее в 18:00.

**Случай 1**

18:00. Пользователи User'!, User2. User3. User4 и User5 в произвольном порядке создают проект, тестовые случаи и наборы.

***Ожидаемый результат 1:***

Предполагается, что каждый пользователь сможет исполнить свои функции без каких-либо заметных отклонений.

**Случай 2**

18:15. Пользователи Userl. User2. User3. User4 и User5 в произвольном порядке вводят результаты тестирования для тестовых случаев и наборов.

***Ожидаемый результат 1:***
Предполагается, что каждый пользователь сможет исполнить свои функции без каких-либо заметных отклонений.

**Случай 3**

18:30. Пользователи User1, User2. User3. User4 и User5 выдают запросы на получение отчетов на уровне проекта, тестового набора и результатов прогона тестовых случаев.

***Ожидаемый результат 3:***
Предполагается, что каждый пользователь сможет исполнить свои функции без каких-либо заметных отклонений.

**Случай 4**

18:45. Пользователи User1, User2. User3. User4 и User5 выводят на принтер результаты запросов.

***Ожидаемый результат 4:***
Предполагается, что каждый пользователь сможет исполнить свои функции без каких-либо заметных отклонений.

Успешное завершение этого теста подтверждает возможность выполнения приведенных выше действий без сбоев в работе системы. Потенциальные отклонения в функционировании системы должны быть описаны дополнительно. Неудовлетворительный результат тестирования свидетельствует о невозможности выполнения указанных функций. Если такая проблема существует, для получения характеристики отказа следует начать с одного пользователя и добавлять каждый раз по одному до тех пор, пока не возникнет системный отказ. Предполагается, что все указанные действия исполняются без каких-либо заметных отклонений.

## Ссылки

*Chris Brown, Test Management Toolkit, Requirements Definition (Набор инструментальных средств управления тестированием, Определение требований).* Документ TMT-RD-10, который размещается под управлением системы контроля документов по адресу:
http://www.tmtcointernal.com/usr/www/docstores/design/requirements/TMT-RD-10.doc
*Chris Brown and J. Barnes, Test Management Toolkit, Test Plan (Набор инструментальных средств управления тестированием, План тестирования).* Документ TMT-TP-10, который размещается под управлением системы контроля документов по адресу:
http://www.tmtcointernal.com/usr/www/docstores/design/plans/TMT-TP-10.doc

## Приложение 1 —Список аббревиатур

API — Application Programming Interface (Интерфейс программирования приложений)
ASCII — American Standard Code for Information Interchange (Американский стандартный код обмена информацией)
CDR — Compact Disc Recordable (Записываемый компакт-диск)
HTML — Hypertext Markup Language (Язык гипертекстовой разметки)
ISO — International Organization for Standardization (Международная организация по стандартизации)
PPC — Power PC (Процессор Power PC)
RISC — Reduced Instruction Set Computing (Сокращенный набор вычислительных инструкций)
SQL — Structured Query Language (Язык структурированных запросов)
SPARC — Scalable Processor Architecture (Масштабируемая процессорная архитектура)
TCL — Tool Command Language (Инструментальный командный язык)
TMT — Test Management Toolkit (Набор инструментальных средств управления тестированием)
X86 — Процессоры серии Intel

## Приложение 2 — Определение терминов

Отсутствует

## Приложение 3 — Сообщения электронной почты от утверждающих лиц

*От: ЧакД. Клут (Chuck D. Klout) [cdklout@tmtco]*

Отправлено: среда 9/11/01 14:23
Кому: Крис Браун (Chris Brown) [cbrown@tmtco]; development@tmtco
Копия: marketing@tmtco; customerssupport@tmtco
Тема: Тестовые процедуры, версия TMT 1.0

Уважаемые члены команды,

Я просмотрел документ спецификации тестовых процедур, TMT-TPS-10, версия 5, и пришел к выводу, что он весьма точно покрывает тестирование требований в данной версии. Я утверждаю этот документ в том виде, в котором он написан. Сокращенный список оборудования и операционных систем, которые будут участвовать в тестировании, обсужден с несколькими заказчиками и утвержден в своей первой версии.

Спасибо, Чак

Чак Д. Клут
Директор отдела маркетинга
TMTCO

*От: Сьюзи Перл (Suzie Perl [spent@tmtco.com]*

Отправлено: четверг 9/12/2001 09.30
Кому: Крис Браун (Chris Brown) [cbrown@tmtco]; development@tmtco
Копия: marketing@tmtco; customersupport@tmtco
Тема: План тестирования TMT 1.0

Привет всем,

Мы вместе с командой просмотрели тестовые процедуры TMT-TPS-10, версия 5, для первой версии приложения TMT и утверждаем их в том виде, в котором они были написаны.

С наилучшими пожеланиями, Сьюзи

Сюзанна Перл
Менеджер, отдел программирования
TMTCO

***От: Брит Гейтер (Bret Gater) [bgater@tmtco.com]***

Отправлено: четверг 9/12/2001 07:30
Кому: Крис Браун [cbrown@tmtco]; test@tmtco; development@tmtco
Копия: marketing@tmtco; costumersupport@tmtco
Тема: План тестирования TMT 1.0

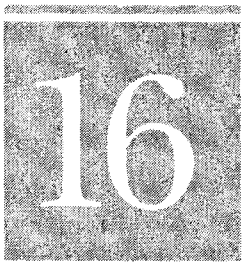
Уважаемые члены команды,

Я просмотрел тестовые процедуры TMT-TPS-10, версия 5, и утверждаю его для использования командой тестеров в том виде, в котором они написаны.

С наилучшими пожеланиями, Брит

Брит Гейтер
Менеджер, отдел программирования
TMTCO

# Пример сводного отчета по системным испытаниям




В главе 5 подробно обсуждался этап системных испытаний процесса разработки продукта. Там же отмечалось, что этот этап выглядит подобно дню соревнований для спортсмена либо дню представления для театрального актера. Большая часть операций по планированию и тестированию уже выполнена, "установлены подмостки" и наступило время действий. На рис. 16.1 показан этап системных испытаний.

В соответствии с рис. 16.1, первой задачей, выполняемой на этапе системных испытаний, является верификация на предмет того, что все подготовлено и можно приступать к выполнению дальнейших действий. Эта задача решается путем применения набора входных критериев системных испытаний, которые были определены в плане тестирования. Если удовлетворены входные критерии, этап системных испытаний может начинаться.

В процессе тестирования выполняется прогон тестов, определенных в плане тестирования. По мере прогона тестов выявляются ошибки, создаются отчеты по ошибкам, а данные, имеющие отношению к процессу отслеживания ошибок, заносятся в специальную базу данных. Между тестировщиками и разработчиками поддерживается непрерывный диалог на основе промежуточных отчетов.

После начала процесса тестирования важно сообщать о состоянии тестирования разработчиками и менеджерам. Подобное общение поддерживается через составление периодических отчетов о состоянии тестирования, а также благодаря подготовке сводного отчета о тестировании в конце фазы системных испытаний. В главе рассматривается пример такого отчета, подготовленного к концу системных испытаний для вымышленного набора инструментальных средств управления тестированием (Test Management Toolkit, TMT).

В главе 5 упоминалось, что назначение сводного отчета о тестировании заключается в ответе на следующие вопросы:

- Что было протестировано?
- Насколько фактические действия по проведению тестированию отклонились от плана тестирования?
- Как график и трудозатраты согласуются с планом тестирования?
- Какие ошибки были найдены?
- Какие ошибки остались на момент завершения тестирования и как они будут обработаны?


***Рис. 16.1. Обзор системных испытаний***

Сводный отчет по тестированию не обязательно включает детализованные результаты прогона каждого теста, однако, он может содержать ссылки на эти результаты в случае, если они были скомпилированы и архивированы. Сбор и архивирование всех результатов тестирования — это одно из преимуществ коммерческого инструментария управления процессом тестирования. Если в результате применения подобного инструментария все тесты объединяются в рамках одного документа, сводный отчет по тестированию будет содержать ссылки на этот документ.

В оставшейся части главы приводится пример сводного отчета по тестированию. Обратите внимание на то, что в этом примере содержатся элементы, которые могут использоваться в других отчетах. Такими элементами, в частности, являются:

- Отчет о состоянии тестирования.
- Отчет об открытых ошибках.
- Сводка по обнаруженным ошибкам по степени их серьезности.

Идентификатор документа: TMT-TSR-10
Версия: 0.2
Авторы: Джеймс Барнс (James Barnes)

### Набор инструментальных средств управления тестированием

### Версия 1.0

# Сводный отчет по тестированию

## Хронология версий

| Версия | Дата | Автор | Описание |
|---|---|---|---|
| 0.1 | 12/02/2001 | Дж. Барнс | Описание результатов тестирования |
| 0.2 | 12/03/2001 | Дж. Барнс | Изменения, основанные на пересмотрах |

## Утверждено

| | | Дата утверждения |
|---|---|---|
| Тестирования | Брит Гейтер (Bret Gater), менеджер отдела тестирования | 12/03/2001 |

## Содержание



### 1. Введение

Назначение этого документа заключается в представлении отчета о результатах системных испытаний набора инструментальных средств управления тестированием (Test Management Toolkit, TMT).
Тесты производились в соответствии с документом "Набор инструментальных средств управления тестированием, План тестирования":

http://www.tmtcointernal.com/usr/www/docstores/desion/Dlans/TMT-TP-10.doc.

Свойства, на которые производятся ссылки в этом документе, находятся в документе определения требований, именуемого "Набор инструментальных средств управления тестированием, Определение требований":

httD://www.tmtcointernal.com/usr/www/docstores/desian/reauirements^MT-RD-10.doc.

### 2. Сводка по результатам тестирования

Три завершенных цикла тестирования, которые были проведены для профаммного продукта TMT, на 100% реализуют комбинированное тестирование системы. К моменту прогона заключительного теста было пройдено 95% тестов, причем катастрофические ошибки обнаружены не были. Используя полученные результаты, команда тестирования рекомендует утвердить выпуск этого программного продукта.

Сводка по ошибкам, остающимся актуальными к концу тестирования, приводится в таблице 2.2. Накопительная сводка по ошибкам, найденным во время тестирования профаммного продукта TMT, показана в таблице 2.3. Сводка по ошибкам также представлена в виде гистограммы на рис. 3.1.

**Таблица 2.1. Отчет о состоянии тестирования**

| Тестовый набор | # тестов | # прошедших | # непрошедших | # не выполненных | % выполненных |
|---|---|---|---|---|---|
| 3.1.1. Пользовательский интерфейс | 20 | **18** | **2** | **0** | 100% |
| 3.1.2. Навигация | **25** | **24** | **1** | **0** | 100% |
| 3.1.3. Аутентификация пользователей — клиент | 12 | 12 | **0** | **0** | 100% |
| 3.1.4. Аутентификация пользователей — администратор | 12 | 12 | **0** | **0** | 100% |
| 3.1.5. Текущие проекты | **15** | **15** | **0** | **0** | 100% |
| 3.1.6. Завершенные проекты | **15** | **15** | **0** | **0** | 100% |
| 3.1.7. Создание проекта | 12 | 12 | **0** | **0** | 100% |
| 3.1.8. Изменение проекта | 12 | 12 | **0** | **0** | 100% |
| 3.1.9. Удаление проекта | **12** | **12** | **0** | **0** | 100% |
| 3.1.10. Создать тестовый случай или набор | **15** | **15** | **0** | **0** | 100% |
| 3.1.11. Измени тестовый случай или набор | **15** | **15** | **0** | **0** | 100% |
| 3.1.12. Удалить тестовый случай или набор | **15** | **15** | **0** | **0** | 100% |
| 3.1.13. Показать тест | 10 | 10 | **0** | **0** | 100% |
| 3.1.14. Показать тестовый набор | **10** | 10 | **0** | **0** | 100% |
| 3.1.15. Прогнать одиночный тест | 10 | **10** | **0** | **0** | 100% |
| 3.1.16. Прогнать тестовый набор | 10 | 10 | **0** | **0** | 100% |
| 3.1.17. Сводный отчет по ошибкам | **12** | 11 | **1** | **0** | 100% |
| 3.1.18. Результаты тестирования/Одиночный тест | **15** | **15** | **0** | **0** | 100% |
| 3.1.19. Результаты тестирования/Тестовый набор | **15** | **15** | **0** | **0** | 100% |
| 3.1.20. Создать матрицу прослеживаемости | 10 | **9** | **1** | **0** | 100% |
| 3.1.21. Резервное копированиеЯестовые случаи | **12** | **12** | **0** | **0** | 100% |
| 3.1.22. Резервное копирование/Тестовые наборы | 12 | 12 | **0** | **0** | 100% |

***Окончание табл. 2.1***

| **Тестовый набор** | **# тестов** | **# прошедших** | **# непрошедших** | **# не выполненных** | **% выполненных** |
|---|---|---|---|---|---|
| 3.1.23. Резервное копирование/Результаты прогона тестов | 12 | 12 | **0** | **0** | 100% |
| 3.1.24. Восстановление/ Тестовые случаи | 12 | 12 | **0** | **0** | 100% |
| 3.1.25. Восстановление/ Тестовые наборы | 12 | 12 | **0** | **0** | 100% |
| 3.1.26. Восстановление/ Результаты прогона тестов | 12 | 12 | **0** | **0** | 100% |
| 3.1.27. Экспорт/Тестовые случаи | 12 | 11 | **1** | **0** | 100% |
| 3.1.28. Экспорт/Тестовые наборы | 12 | 11 | **1** | **0** | 100% |
| 3.1.29. Экспорт/Результаты **прогона тестов** | 12 | 11 | **1** | **0** | 100% |
| 3.1.30. Справка | **18** | **18** | **0** | **0** | 100% |
| 3.1.31. Многопользовательские функциональные возможности | 10 | **10** | **0** | **0** | 100% |
| | 408 | 400 | **8** | **0** | 100% |

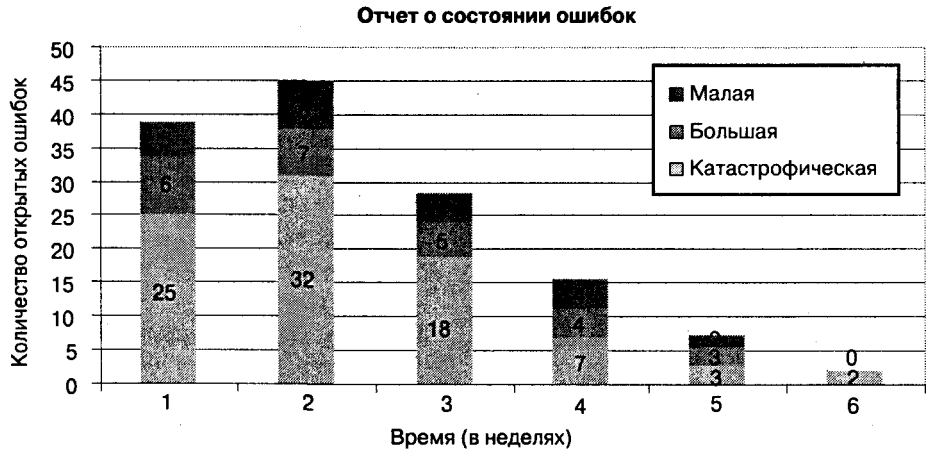



***Рис. 3.1. Сводка по ошибкам разной степени серьезности***

**Таблица 2.2. Отчет об открытых ошибках**

| **Инструментарий по управлению тестированием<br>Сводка по открытым ошибках** | | | | | |
|---|---|---|---|---|---|
| Дата просмотра: 30-11-02 | | | | | |
| Посетители | | | | | |
| **Идентификатор ошибки** | **Состояние** | Серьезность | **Дата обнаружения** | **Сводка по ошибке** | **Действие** |
| B3.1.1A | Назначено | Малая | 15-10-01 | Ошибка в метке поля "Test Identifier". Tset Identifier | Исправлено в 1.0 |
| B3.1.1B | Назначено | Малая | 16-10-01 | Ошибка в метке поля "Test Name". *Test Nnme | Исправлено в 1.0 |
| B3.1.2 | Назначено | Малая | 17-10-01 | Пропущена метка меню "Utility Menu" | Исправлено в 1.0 |
| B3.1.16 | Назначено | Малая | 20-10-01 | Наложение полей Pass (Пройдено) и Fail (Сбой) | Исправлено в 1.0 |
| B3.1.19 | Назначено | Малая | 22-10-0.1 | Попытка выбора матрицы трассировки в корневом каталоге. Должен быть каталог /Trace | Исправлено в 1.0 |
| B3.1.26 | Назначено | Большая | 01-11-01 | При экспорте "Test Case" (Тестовый случай) данные оказываются разрушенными | Отложено до версии 1.1 |
| B3.1.27 | Назначено | Большая | 01-11-01 | При экспорте "Test Suite" (Тестовый набор) данные оказываются разрушенными | Отложено до версии 1.1 |
| B3.1.28 | Назначено | Большая | 01-11-01 | При экспорте "Test Result" (Результаты прогона тестов) данные оказываются разрушенными | Отложено до версии 1.1 |
| Примечания: B3.1.26, B3.1.27 и B3.1.28 — для решения проблемы требуются архитектурные изменения, которые будут выполнены в следующей версии программы | | | | | |

**Таблица 2.3. Сводка по ошибкам, найденным во время тестирования**

| | **Неделя 1** | **Неделя 2** | **Неделя 3** | **Неделя 4** | **Неделя 5** | **Неделя 6** |
|---|---|---|---|---|---|---|
| Катастрофические | **5** | **6** | **4** | **4** | **2** | **0** |
| Крупные | **6** | **7** | **5** | **4** | **3** | **0** |
| Мелкие | **25** | **32** | **18** | **7** | **3** | **2** |

## 3. Свойства, которые должны тестироваться

С целью обеспечения гарантии того, что программный продукт TMT удовлетворяет требованиям, указанным в спецификации требований TMT, были протестированы следующие свойства:

- Требование 3.1.1. Пользовательский интерфейс
- Требование 3.1.2. Навигация
- Требование 3.1.3. Аутентификация пользователей — клиент
- Требование 3.1.4. Аутентификация пользователей — администратор
- Требование 3.1.5. Текущие проекты
- Требование 3.1.6. Завершенные проекты
- Требование 3.1.7. Создание нового проекта
- Требование 3.1.8. Изменение проекта
- Требование 3.1.9. Удаление проекта
- Требование 3.1.10. Создание тестового случая или набора
- Требование 3.1.11. Изменение тестового случая или набора
- Требование 3.1.12. Удаление тестового случая или набора
- Требование 3.1.13. Отображение теста
- Требование 3.1.14. Отображение тестового набора
- Требование 3.1.15. Прогон одиночного теста
- Требование 3.1.16. Прогон тестового набора
- Требование 3.1.17. Создание списка прогона
- Требование 3.1.18. Выполнение списка прогона
- Требование 3.1.19. Сводный отчет по ошибкам
- Требование 3.1.20. Результаты тестирования — одиночный тест
- Требование 3.1.21. Результаты тестирования — тестовый набор или список прогона
- Требование 3.1.22. Создание матрицы прослеживаемости
- Требование 3.1.23. Резервное копирование тестовых случаев
- Требование 3.1.24. Резервное копирование тестовых наборов
- Требование 3.1.25. Резервное копирование результатов прогона тестов
- Требование 3.1.26. Восстановление тестовых случаев
- Требование 3.1.27. Восстановление тестовых наборов
- Требование 3.1.28. Восстановление результатов прогона тестов
- Требование 3.1.29. Экспорт тестовых случаев
- Требование 3.1.30. Экспорт тестовых наборов
- Требование 3.1.31. Экспорт результатов прогона тестов
- Требование 3.1.32. Получение справки
- Требование 3.1.33. Многопользовательские функциональные возможности
- В ходе прогона тестов по всем свойствам получены результаты, которые описаны ранее, в разделе 2.

## 4. Свойства, которые не должны тестироваться

Ниже приводится список функциональных свойств и/или конфигураций системы, которые не тестировались.

- Web-сервер (Apache или IIS) непосредственно не тестировался.

- Не проводилось расширенное тестирование клиент/серверной архитектуры под большой нагрузкой. Многопользовательские функциональные возможности тестировались для пяти реальных пользователей, что является минимальной многопользовательской конфигурацией.

## 5. Отклонения от плана тестирования

Отклонения от плана тестирования не обнаружены.

## 6. Соблюдение графика

Тестирование было завершено по графику, хотя были обнаружены небольшие отклонения дат начала и завершения для индивидуальных циклов тестирования.

## 7. Ссылки

*Chris Brown, Test Management Toolkit, Requirements Definition (Набор инструментальных средств управления тестированием, Определение требований).* Документ TMT-RD-10, который размещается под управлением системы контроля документов по адресу:

http://www.trntcointernal.corn/usr/www/docstores/clesiqn/reauirernents/TMT-RD-10.doc

*Chris Brown and J. Barnes, Test Management Toolkit, Test Plan (Набор инструментальных средств управления тестированием, План тестирования).* Документ TMT-TP-10, который размещается под управлением системы контроля документов по адресу:

httD://www.tmtcointernal.com/usr/www/docstores/design/plans/TMT-TP-10.doc

*Chris Brown и J. Barnes, Test Management Toolkit, Release 1.0. Test Procedure Specification (Набор инструментальных средств управления тестированием, Спецификация тестовой процедуры TMT).* Документ TMT-TPS-10, который размещается под управлением системы контроля документов по адресу:

httD://www.tmtcointernal.com/usrAwww/docstores/design/plans^MT-TPS-10.doc

## Приложение 1 —Список аббревиатур

Отсутствует

## Приложение 2 — Определение терминов

Отсутствует

## Приложение 3 — Сообщения электронной почты от утверждающих лиц

***От: Брит Гейтер (Bret Gater) [bgater@tmtco.com]***

Отправлено: четверг 9/10/2001 07:30
Кому: Дж. Варне [jbarnes@tmtco]; test@tmtco; development@tmtco
Копия: marketing@tmtco; costumersupport@tmtco
Тема: Сводка по тестированию TMT 1.0

Уважаемые члены команды,

Я просмотрел сводный отчет по тестированию TMT-TSR-10, версия 2, и утвердил его, поскольку он точно представляет результаты тестирования. Команда тестирования рекомендует утвердить выпуск программного продукта TMT, версия 1.0. Поздравляю разработчиков и тестировщиков с достигнутым успехом!

С наилучшими пожеланиями, Брит

Брит Гейтер
Менеджер, отдел программирования
TMTCO

# Литература


1. Albrecht, Allan J. (1979). "Measuring Application Development Productivity." *Proceedings of the IBM Application Development Symposium,* 83-92.
2. Basili, V. R. and D. M. Weiss. (1984). "A Methodology for Collecting Valid Software Engineering Data." *IEEE Transactions on Software Engineering, SE-10* (6).
3. Beizer, Boris. (1995). *Black-Box Testing: Techniques for Functional Testing of Software and Systems.* New York: Wiley.
4. Belford, P. C, R. A. Berg, and T. L. Hannan. (1979). "Central Flow Control Software Development: A Case Study of the Effectiveness of Software Engineering Techniques," Proceedings from the Fourth Summer Software Engineering Workshop, SEL-79-005.
5. Black, Rex. (1999). *Managing the Test Process.* Redford, WA: Microsoft Press.
6. Boehm, Barry W. (1981). *Software engineering economics.* Englewood Cliffs, NJ: Prentice-Hall, Inc.
7. Boehm, Barry W. (1988). "A spiral model for software development and enhancement." *IEEE Computer,* 21(5) (May): 61-72.
8. Boehm, Barry W. (1991). "Software risk management: Principles and practices." *IEEE Software,* 8(1) (January): 32-41.
9. Boehm, Barry W. (2000). *Software Cost Estimation with COCOMO II.* Englewood Cliffs, NJ: Prentice Hall.
10. Brooks, Fred. (1982). *The Mythical Man-Month: Essays on Software Engineering.* Reading, MA: Addison-Wesley.
11. Burnstein, Ilene, C.R. Carlson, and T. Suwanassart. (1996). "Developing a Testing Maturity Model." *Proceeding of the Ninth International Software Quality Week Conference,* San Francisco, May, 1996. *Note:* Ilene Burnstein is affiliated with the Illinois Institute of Technology, where work on the Testing Maturity Model is being conducted.
12. Cooley, J. W., and J. W. Tukey. (1965). "An Algorithm for Machine Calculation of Complex Fourier Series." *Mathematics of Computation,* Vol. 19, pp. 297-301.
13. Curtis, Bill, Herb Krasner, Vincent Shen, and Neil Iscoe. (1987). "On building software process models under the lamppost." *Proceedings of the 9th International Conference on Software Engineering* (pp. 96-103). Monterey, CA: IEEE Computer Press Society.
14. DeMarco, Tom, and Timothy Lister. (1987). *Peopleware: Productive Projects and Teams.* New York: Dorset House Publishing.
15. Dustin, Elfriede, Jeff Rashka, and John Paul. (1999). *Automated Software Testing: Introduction, Management, and Performance.* Reading, MA: Addison-Wesley.
16. Fagan, M.E. (1976). "Design and code inspections to reduce errors in program development." *IBM Systems Journal,* 15(3): 182-210.
17. Fewster, Mark, and Dorothy Graham. (1999). *Software Test Automation.* Reading, MA: Addison-Wesley.
18. Good, Donald I., R. M. Cohen, C G. Hoch, L. W. Hunter, and D. F. Hare. (1978). "Certifiable Minicomputer Project, ICSCA," Report on the Language Gypsy, Version 2.0. Technical Report ICSCA-CMP-10, The University of Texas at Austin, September 1978.
19. Humphrey, Watts S. (1990). *Managing the Software Process.* Reading, MA: Addison-Wesley.

20. Humphrey, Watts S. (1997). *Introduction to the Personal Software Process.* Reading, MA: Addison-Wesley.

21. Humphrey, Watts S. (2000). *Introduction to the Team Software Process.* Reading.MA: Addison-Wesley.

22. IEEE. (1983). *IEEE Standard 829: IEEE Standard for Software Test Documentation.* Los Alamitos, CA: IEEE Computer Society Press.

23. IEEE. (1984). *IEEE Standard 830: The IEEE Guide to Software Requirements Specifications.* Los Alamitos, CA: IEEE Computer Society Press.

24. IEEE. (1993). IEEE Standard 1044, *IEEE Standard for Software Anomalies,* © 1993 IEEE, New York, NY.

25. Jones, Capers. (1986). *Programming Productivity.* New York: McGraw-Hill.

26. Jones, Capers. (1997). *Software Quality: Analysis and Guidelines for Success.* Boston: International Thompson Computer Press.

27. Kaner, Cem, Jack Falk, and Hung Quoc Nguyen. (1999). *Testing Computer Software* (2nd ed.). New York: Wiley.

28. Kit, Edward. (1995). *Software Testing in the Real World: Improving the Process.* Reading, MA: Addison-Wesley.

29. Koomen, Tim, and Martin Pol. (1999). *Test Process Improvement.* Reading, MA: Addison-Wesley.

30. Lewis, William E. (2000). *Software Testing and Continuous Quality Improvement.* Boca Raton, FL: Auerbach.

31. McCabe, Thomas J. (1976). "A Complexity Measure." *IEEE Transactions on Software Engineering.*

32. McCabe, Thomas, and Charles W. Butler. (1989). "Design Complexity Measurement and Testing." *Communications of the ACM 32,* 12 (December 1989): 1415-1425.

33. McConnell, Steve. (1996). *Rapid Development: Taming Wild Software Schedules.* Redmond, WA: Microsoft Press.

34. Michael Fagan. (1976). "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal, 15 (Ъ),* 182-211.

35. Musa, John D. (1993). "Operational Profiles in Software-Reliability Engineering." *IEEE Software,* 14-19.

36. Myers, Glen. (1979). *The Art of Software Testing.* New York: Wiley.

37. Myers, GlenfordJ. (1977). "An Extension to the Cyclomatic Measure of Program Complexity." *SIGPLAN Notices.*

38. Paulk, Mark, Charles Weber, and Bill Curtis. (1995). *The Capability Maturity Model: Guidelines for Improving the Software Process.* Reading, MA: Addison-Wesley.

39. Paulk, Mark. "Using the Software CMM with Small Projects and Small Organizations." In Eugene McGuire (1999), *Software Process Improvement: Concepts and Practices.* Hershey, PA: Idea Group Publishing.

40. Perry, William E. (2000). *Effective Methods for Software Testing(2nd* ed.). New York: Wiley.

41. Perry, William E., and Randall W. Rice. (1997). *Surviving the Top Ten Challenges of Software Testing.* New York: Dorset House Publishing.

42. Pfleeger, Shari Lawrence. (2001). *Software Engineering: Theory and Practice* (2nd ed.). Upper Saddle River, NJ: Prentice Hall.

43. Pressman, Roger. (1997). *Software Engineering: A Practitioner's Approach* (4th ed.). New York: McGraw-Hill.

44. Robertson, Suzanne, and James Robertson. (1999). *Mastering the Requirements Process.* Reading, MA: Addison-Wesley.

45. Schulmeyer, G. Gordon and Garth R. Mackenzie. (2000). *Verification and Validation of Modern Software-Intensive Systems.* Upper Saddle River, NJ: Prentice Hall.

46. Software Productivity Consortium (1993). Software Error Estimation Program (SWEEP) User Manual, (SPC-92017-CMC), Version 02.00.10.

47. Sommerville, Ian. (1992). *Software Engineering* (4th ed.). Reading, MA: Addison-Wesley.

48. The Standish Group. (1994). *The CHAOS Report.* Dennis, MA: The Standish Group.

49. The Standish Group. (1995). *The Scope of Software Development Project Failures.* Dennis, MA: The Standish Group.

50. van Soligen, Rini, and Egon Berghout. (1999). *The Goal/Question/Metric Method: A practical guide for quality improvement of software development.* Berkshire, England: McGraw-Hill Book Company, UK.

51. *p.* 5 From *Verification and Validation of Modern Software-Intensive Systems* by Schulmeyer *&c* MacKenzie. © 2000. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ.

52. *pp. 6~7, 73* From *Rapid Development: Taming Wild Software Schedules* by Steve McConnell. Copyright 1996. Reproduced by permission of Microsoft Press. All rights reserved.

53. *pp. 10, 40* From *Software Engineering: Theory and Practice,* 2nd ed., by S. Pfleeger. © 2001. Material is reprinted by permission of Pearson Education, Inc.

54. *pp. 25, 66, 134, 139* From *Software Engineering Economics* by Boehm, B.W. © 1981. Reprinted by permission of Pearson Education, Inc., Upper Saddle River, NJ.

55. *p. 28* From *Software Engineering: A Practitioner's Approach,* 4th ed., by R. Pressman. © 1997 McGraw-Hill, Inc.

56. *p. 31* Information regarding the content and format of requirements documents is reprinted with permission from IEEE Std. 830-1993: *The IEEE Guide to Software Requirements Specifications.* Copyright 1993 by IEEE.

57. *p. 36* From *Software Testing in the Real World: Improving the Process,* by E. Kit. © 1995 Addison-Wesley, Inc.

58. *p.* 55 From *Automated Software Testing (pp.* 32-38) by E. Dustin, J. Rashka, &J. Paul. © 1999 Addison Wesley Longman, Inc. Reprinted by permission of Pearson Education, Inc.

59. *p. 61* Material excerpted from *Managing the Test Process* by permission of the author, Rex Black; second edition in press by John Wiley & Sons, Inc., ISBN 0-471-22398-0.

60. *p. 70* From *The Mythical Man-Month* (p. 18) by F. Brooks, Jr. © 1999 Addison Wesley Longman, Inc. Reprinted by permission of Pearson Education, Inc.

61. *pp. 77,* SO Information regarding the content and format of test documents is reprinted with permission from IEEE Std. 829-1983: *IEEE Standard for Software Test Documentation.* Copyright 1983 by IEEE.

62. *pp. 214-215* Information regarding defect reporting is reprinted with permission from IEEE Std. 1044-1993: *IEEE Standardfor Software Anomalies,* Copyright 1993 by IEEE.